* JAVA *

03 Feb 2021

① Class decleration

② main method

③ Printing statement

→ Access modifier class clsname

↓

Public
Private
Protected
default

(:)

↓

arguments (or) Parameters

Access modifier

non access modifier → static, non static, final, abstract

return type

```
{
Public static void main (String args[])
{
System.out.println ("This is my sample program");
}
}
```

method name

Public → Anyone can access freely

static → Accessable without object

void → No specific return type

System → Class predefine

out → Object

println → method (Predefine non static method)

PROGRAM:-

A set of instructions in a programming language is called program.

There are three important parts, they are

1) Class decleration

2)

=> Every program in java starts with a class

=> Class is followed by classname.

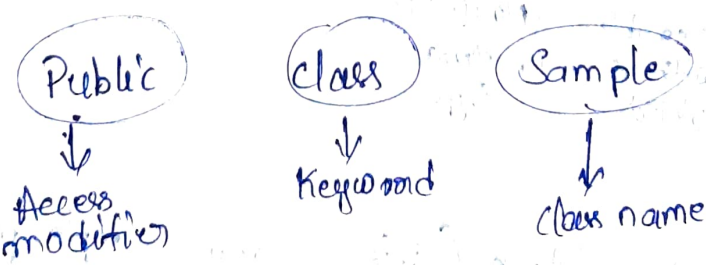
=> class is a keyword (reserved word) which have some reserve meaning.

=> Classname is a combination of [A-Z, a-z, 0-9, $, _]

=> classname for a standard should start with a capital letter

If classname has more than one word spaces are not allowed.

Ex:



Secoud part of a program:-

The most imp part of every java program

2) main method.

It is return as

Public static void main (String args[])

* public → Access modifier
  It indicates that main method is accessible freely.
* static → Non access modifier
  It indicates method is accessable without object creation. (static means accessable without object and non-static means accessable with object).
* void → it is a return type
  It indicates there is no specific return type, &
* main → it is a name of a method
* String args[] → Command line arguments

NOTE :- main method syntax is very strict ie, we are not supposed to change single word.

Third Part :- (Printing statement)

If we want to print something on the screen we have to use

System.out.println(" ");

* System → It is a class (predefine)
* Out → It is an object (predefine)
* println / print → It is a method (predefine)

CONCLUSION :-

We want to access println method since it is non static we are accessing through "out" object which is present in System class

NOTE :- In java there is no word called as default ie, if we did not write either public, private & protected then it is considered as default.

In java there is no word called as non static ie, if we did not write static, final & abstract then it is considered as non static.

atement)

ething on the screen

");

ne define)

fine)

method (predefine)

u method since it
accessing through "out"
in System class

word called as
t write either
the it is

word called as
did not write
then it is
tic.

```
class MyInfo
{
  public static void main (String args[])
  {
    System.out.print ("Qspiders");
    System.out.println ("2019");
    System.out.print ("65%");
    System.out.println ("CSE");
  }
}
```

In the above program, first JVM will print Qspiders and then it will check whether it is print or println.

If it is print then next statement will be printed in same line.

If it is println, next statement will be printed in new line.

Therefore we can conclude that print or println will not effect to current statement, rather it effects to next statement.

Main method syntax is very strict ie., we can't change it but still there are some changes which are acceptable.

Output:
Qspiders 2019
65% CSE

They are

=> We can change the place of modifiers.
ie., public static <---> static public

=> Command like arguments [] ie., an array can be written in three ways
1) String args[]
2) String[] args
3) []String args

** Can we write main method as private?

=> If we write main method as private, our program will compile successfully.

Our program will compile successfully but it will not be executed because if the method is private it is not available or accessable directly from outside.

If we want program to be executed JVM has needs to access main method outside.

* Can we keep main method as non static?

~~If we keep main method as non static we will not get any compile time error~~

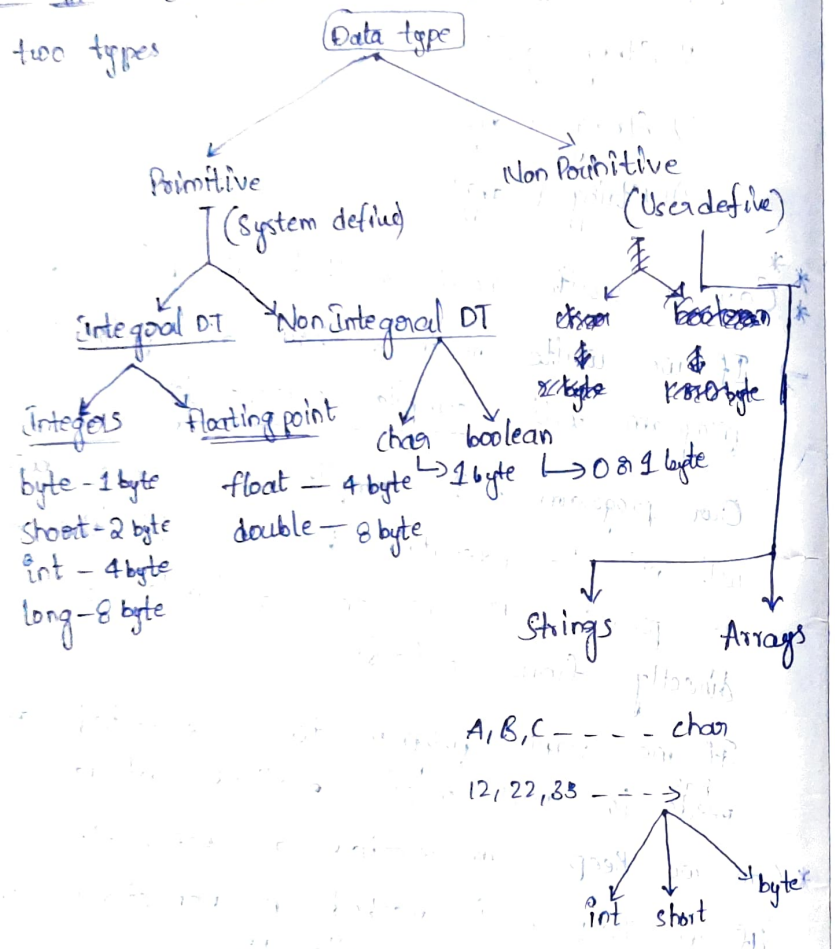
=> If we keep method as non static there is no compile time error but program will not be executed because method is non static JVM has to create an object and then access it.

Data:- It is defined as information

Ex:- Name of a person
Age
Conduct etc.

Data type:- It defines the type of data.

two types



A, B, C - - - - char

12, 22, 33 - - - →

789345G, 2812356 - - → long

0.23, 0.24, 0.33 → float / double

true, false → boolean

NOTE: Here we have to remember the size and when to use which data type.

All the primitive data type are like keywords so they all will be in smaller case.

char → means only one character

If it is more than one character we have to define type as string.

Variables:- Used to store the data

for using variable we have to follow two steps → 1) Variable declaration
2) Variable initialisation

① **Variable declaration**

Syntax is `Datatype Variablename;`

Ex:- int i;
float b;
char c;
String s;

② **Variable Initialisation**

Syntax is `Variablename = value;`

Ex:- i = 100;
b = 0.23f; → mandatory to indicate float
c = 'A'; → to indicate character
s = "java";

⇒ Variable name can be a combination of

A-Z
a-z
0-100
$; , -

⇒ Whenever we declared a variable there will be one memory block that gets created.

⇒ Whenever we initialise a value that value get stored in that memory block.

⇒ We can declare and initialise a variable in single statement also.

Syntax is `Datatype varname = value;`

Ex: int a = 100;

✓ int a = 10, b = 20;

✓ int a = 10, b = 20, c;

✗ char C = 'AB'; → (Only 1 character should be)

✓ String s = "java", s1 = "sql";

✓ String s = "123";
✓ String s1 = " ";
✓ String s2 = "A";
} we can represent int, char or float also values as string also by keeping " " → Space

✓ double d = 100; → 100.0

✗ float f1 = 22.27; → f missing

✗ int i = 100.0;

Syntax for printing the data stored in a variable is: `System.out.println(varname);` (a)

System.out.println (b);

€ Don't print 'a' rather print value stored in a & b

PROGRAM :-

Write a program to print book name, book price, book rating using datatypes and variables.

```
Public class bookdetails
{
  public static void main(String[] args)
  {
    String bookname = " JAVA";
    int bookprice = 500;
    char rating = 'A';
    System.out.println (bookname);
    System.out.println (bookprice);
    System.out.println (rating);
  }
```

O/P :- JAVA
500
A

Write a program to print studentname, age, percentage, height, year of passout and char name using datatypes and variables

public

Assignment operator (=) :-

It is used to assign or store the data into a variable

Ex:-
```
int a = 10;
int b = 20;
a = a+b;
```

Comparison operator (==) :-

It is used to compare two values.

It is a boolean operator ie; if the numbers are same it gives the output as true else output as false.

Ex:- Sop(10==10) -> true
(== : boolean)

```
class sample
{
  public static void main(String args[])
  {
    int a = 10, b = 10;
    Sop(a==b);
  }
}
```

Program :-

```
class sample
{
    public static void main(String args[])
    {
        int a = 10, b = 20;
        Sop(a==b);
        int c=30,d=30;
        Sop(c==d);
// int e=(a==b);
// Sop(e);
        boolean f=(a==b);
        Sop(f);
    }
}
```

Expected: int
found: boolean

a=10, b=20 → false

Aiorthematic Operations :-

It is used to perform all ~~aiorthematic~~ aiorthematic operations like

- `+` ⟶ addition
- `-` ⟶ Subtraction
- `*` ⟶ multiplication
- `/` ⟶ division ⟶ (coefficient)
- `%` ⟶ mode ⟶ (remainder)

Write a program to perform all aiorthematic operations on a given two numbers ie; 10 & 2

```
class AiorthOperations
{
    public static void main(String args[])
    {
        int a=10, b=2, c, d, e, f, g;
        c=a+b; // 10+2 = 12 c
        d=a-b;
        e=a*b;
        f=a/b;
        g=a%b; // 10%2 = 0 g
        So.plu(c);
        Sop lu(d);
        Sop lu(e)
        Sop lu(f);
        Sop lu(g);
```

07/02/2020

Operator overloading :-

* One operator behaving in multiple ways is called as operator overloading.
* Java doesn't support operator overloading but there is one exceptional case where operator overloading is occuring. ie, '+' operator.

Addition

int + int = int
10 + 20 = 30

int + char = int
10 + 'A' (65) = 165

char + char = int
'A' (65) + 'A' (65) = 130

int + float = float
100 + 100.2f = 200.2f

long + double = double
172727 + 172727.2525 =

byte + int = int
10 + 100 = 110

Concatination

String + class = String
"JAVA" + 'A' = JAVAA

String + int = String
"JAVA" + 122 = JAVA122

String + long = String
"JAVA" + 12722 = JAVA12722

String + float = String
"JAVA" + 0.2f = JAVA0.2f

String + byte = String
"JAVA" + 10 = JAVA10

String + String = String
"JAVA" + "91" = JAVA91

Unicodes

A	B	C	D	E	.....	Z
65	66	67	68	69		90

a	b	c	d	e	.....	z
97	98	99	100	101		

Conclusion :-

⇒ When we are adding two integral datatypes, two floating datatypes, character with integral datatypes and char with floating datatype '+' operator will always act as addition.

Whenever we add a char JVM will take the unicode values and they are

A, B, C, ..., Z & a, b, c, ... z
65 66 67 90 97 98 99

⇒ Out of two operant if anyone operant is string '+' operator will always act as concatination and result will always be a string.

fx

PROGRAM

```
class sample
{
  public static void main (String args[])
  {
    int a=20, b=19;
    char ch='A';
    String s="JAVA", s1="Sql";
    Sopln(a+b); // 20+20 --> 40
    Sopln(a+ch); // 20+65 --> 85
    Soplu(ch+ch); // A+A --> 130
    Soplu(a+s); // 20+JAVA --> 20JAVA
    Soplu(s+s1); // JAVA+Sql --> JAVASql
    Soplu(s+ch); // JAVA+A --> JAVAA
    Sop("bookname is"+s); // bookname is JAVA
```

```
Sop(a+c+b+ch); // 20 JAVA 20A
    }
}
```

Output :-

```
30
75
130
10 JAVA
JAVA Sql
JAVA A
bookname is JAVA
10JAVA20A
```

Ex PROGRAM

```
{
String bname = "Java";
  int bookprice = 700;
Sop ("name of book is :" + bname);
Sop ("Price is " + bookprice + "/-");
Sop (" name is:" + bname + "\n" +
     " price is :" + bookprice + "Rs/-");
}
}
```

Relational Operators :-

It defines the relationship b/w two parameters

- `>` → Greater than
  - 10>20 ✓ → false
  - 'a' > 'b' ✓ → false
  - true > false ✗ (invalid)
  - 10<20 ✓ → true Java > Sql (invalid) ✗
- `<` → lesser than
  - 10<20 → true
  - 'z' < 'a' → true (90 < 97)
- `>=` → Greater than equals
  - 10>=20 ✓
  - 'a' >= 'b' ✓
- `<=` → lesser than equals
  - 10<=20 ✓
  - 'a' <= 'b' ✓
- `!=` → not equals
  - 10 != 20 ✓
  - a != 'A' ✓

Ex:- 20 > 10 < 30 → invalid

20 > 10 → true

true < 30 → invalid

> Since it is true which is boolean. We can't say boolean less than int

Logical Operators :-

When we want to check more than one condition we can use logical operators.

Not (!)

0	1
1	0

AND (&&)

0	0	0
0	1	0
1	0	0
1	1	1

OR (||)

0	0	0
0	1	1
1	0	1
1	1	1

Ex: 1) ((10>20) && (20<30)) → false
(false, true)

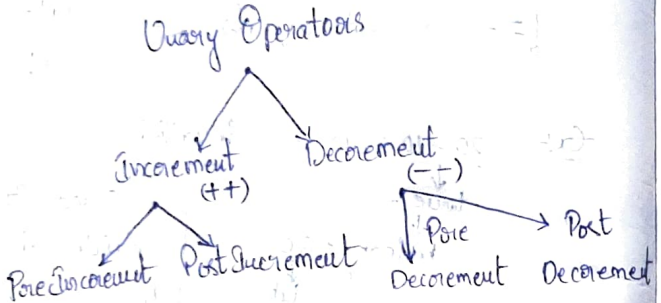
2) ((10>20) || (20<30)) → true
(false, true)

3) (10 != 20) → true

4) (10 == 20) → false

08/02/2020

Unary Operators:



Increement:

Increasing the value by "one", denoted by (++).

Increement is again divided into

* Pre Increement

Rule:- first increment the value then assign or print it

It is denoted as ++ Variablename

Ex:-
```
class sample
{
    psvm( )
    {
        int a = 10;
        int b = ++a;
        Sop(b);
    }
}
```

O/P :- 11

* Post Increement

Rule:- first assign the value then increment it.

It is denoted as "variablename ++"

Ex:-
```
class sample
{
    psvm( ):
    {
        int a = 10;
        int b = a++;
        Sop(b); // 10
        Sop(a); // 11
    }
}
```

O/P :- 10
11

Decrement :

Decreasing the value by 'one', denoted by (--).

It is divided into two types.

* Pre Decrement

Rule: first decrement the value then assign and print it.

It is denoted by "-- varname"

Ex:-
```
class sample
{
  p s v m ( ) 0
  {
    int a = 10;
    int b = --10;
    Sop(b);
  }
}
```

o/p :- 9

* Post Decrement

Rule- first assign the value and then decrement it.

Denoted as varname'--'

Ex:-
```
class same
{
  p s v m ( )
  {
    int a = 10;
    int b = a--;
    Sop (b) // 10
    Sop (a) // 9
```

o/p :- 10
9

Expression	Initial value of 'x'	Value of 'y'	final value of 'x'
y = x++	10	10	11
y = ++x	10	11	11
y = x--	10	10	9
y = --x	10	9	9

Ex:- 1

int a = 100; 101 102 103

Sop(a++ + ++a + a++);
(a++: 101 → 100; ++a: 102 → 102; a++: 103 → 102)

o/p :- 304

Ex:-2

int b = 200;

Sop(--b + b-- + ++b + b++);
(--b: 199 → 199; b--: 198 → 199; ++b: 199 → 199; b++: 199 → 199)

o/p :- 796

Ex:-3 int c = 550; 549 548 547

Sop(--c - c-- + c++ - c-- + c--);

549 ↓ 549 - 549 ↓ + 548 ↓ - 548 ↓ + 547 ↓

O/P :- ~~27888~~ 547

10feb2020

IDENTIFIERS :-

Names given by the programmer as per convention.

Ex:- classname, varname, methodname, packagename and projectname.

Rules for defining identifiers

1) Usually clsname starts with capital letter and it is the combination of A-Z, a-z, 0-9, $, _

Ex:- class Sample ✓
class S$ample ✓
class Sample_1 ✓
class sample ✗
class s#ample ✗

2) Usually varname starts with a small letter case and it is a combination of A-Z, a-z, 0-9, $, _

Ex:- int age1;
float percent_1yr;
String name; ✗

Rule

3) If a classname & varname has more than one word then spaces are not allowed b/w the words.

Ex:- class Sample Program ✗
class SampleProg ✓
int age of pavan; ✗
int Pavanage ✓

4) For Classname & variablename cannot start with a digit.

Ex:- * class 1Sample ✗
int 10age ✗
class S1ample ✓

5) clsname & varname cannot be a keyword.

Ex:- class int
int if;
class _Class

6) clsname & varname can be a name of predefined classes and interface. But it is not recommended to use.

Ex:- class Runnable → Predefined interface
✓ int String → Predefine class

Examples:

int $var; ✓

float Percentage-1; ✓

but not recommended — String String; ✓

✗ not allowed — class Sample ✗

can't start with digit — byte 1Size; ✗

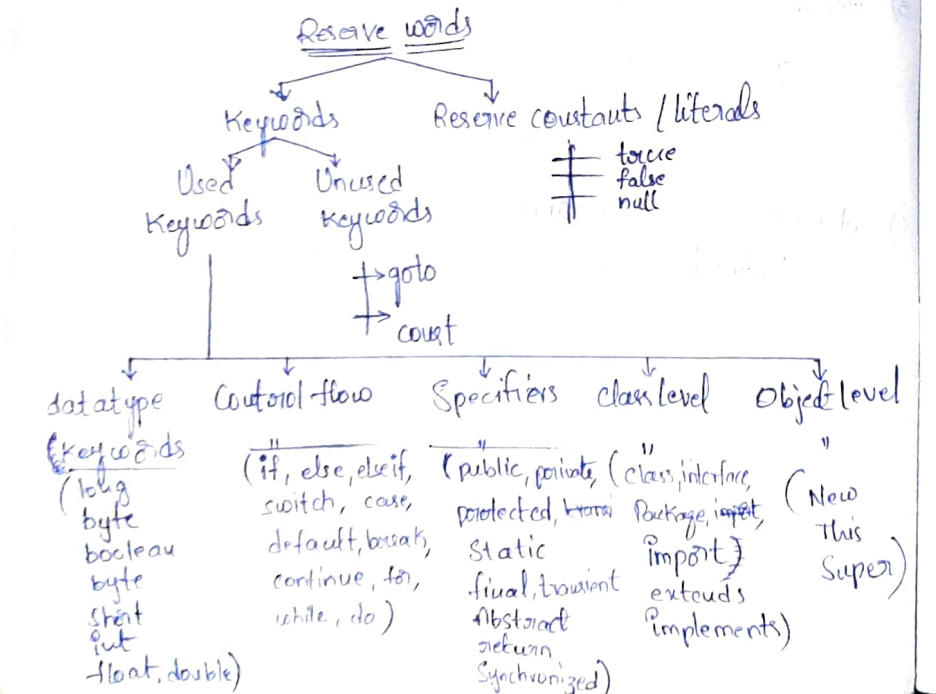
spaces not allowed — class First Program ✗

char _var; ✓

KEYWORDS :-

Keywords are the reserved words which have some reserve meaning.

All the keywords should must be starts with smaller case.

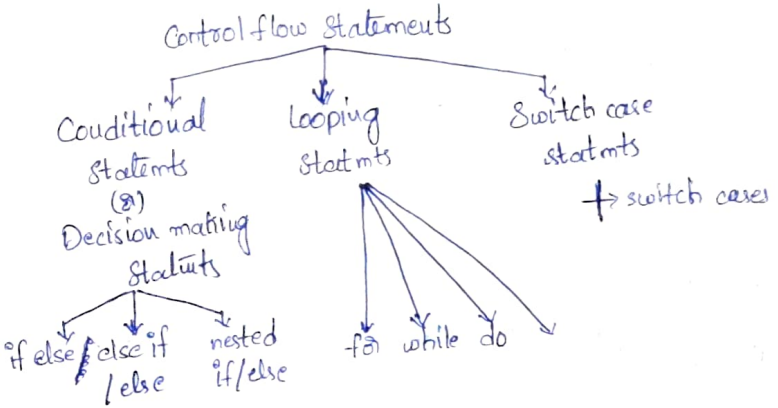


Exception Keywords

(try
catch
throw
throws
finally)

Control flow statements :-

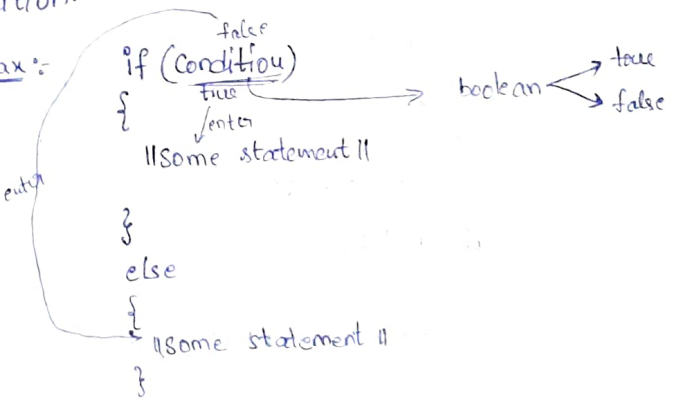
It controls the flow of execution and they are;



Conditional statements (or) Decision making :-

It changes the flow of execution depending on condition.

Syntax :-



Ex:

① `if (1)` ← not boolean
× {
}

② `if (10==20)`
✓ {
}

③ `if (10>20)`
✓ {
}

④ `if (10!=20)`
✓ {
}

⑤ `if ('a'>'b')`
✓ {
}

```
else
{
}
```

PROGRAMS:

1) Greatest of two no's 10 & 20

```
public class sample
{
  public static void main(String args[])
  {
    int a=10, b=20;
    if (a>b);
    if (a>b)
    {
      Sop(a+" is greater of two nos");
    }
    else
    {
      Sop(b+" is greater of two no's");
    }
  }
}
```

2) WAP to check Raju is eligible to vote or not

Prog

```
class sample
{
  public static void main(String[] args)
  {
    int age=23;
    if (age>18)
    {
      Sop(raju is eligible to vote");
    }
    else
    {
      Sop(".. .. not eligible");
    }
  }
}
```

```
String name="Raju";
if (eligible to vote)
{
  Sop("name + eligible to vote");
}
else
{
  Sop("name + not eligible to vote");
}
```

3) WAP to check whether number ie 15 is divisible by 2 or not

4) WAP to check 22 is an even number or odd number

3)

```
public class sample
{
 public static void main(String args[])
{
Int num = 15;
if (num%2 == 0)
{
 Sop(num + "divisible by 2");
}
else
{
 Sop(num+ "not divisible by 2");
}
}
```

4)

```
public class sample
{
 public static void main(String args[])
 {
  int a
  int atm = 22;
  if (22%2 == 0)
  {
   Sop(a+ "is an even number");
  }
 else
  {
   Sop(a + "is an odd number");
  }
 }
}
```

5) WAP to check whether number is +ve & -ve

```
public class sample
{
 public static void main (String args[])
 {
  int a = 17;
  if(a%2 == 0)
  {
   Sop(a + "is a positive number")
```

```
int num = 30;
if(num>=0)
{
  Sop("Number is positive");
}
else
{
  Sop("num"is negative");
}
}
}
```

TYPE2 :- Syntax

```
class Sample
{
  p s v m(String args[])
  {
    if(Condition 1  logical operator  Condition 2)
    {
      // Some statement //
    }
    else
    {
      // Some statement //
    }
  }
}
```

Ex:- Program

WAP to check couple is eligible to get married or not

```
public class Couplemrge
{
  public static void main(String args[])
  {
    int b = 21, g = 18;
    if (b >= 21 && g >= 18)
    {
      Sop("eligible");
    }
    else
    {
      Sop("not eligible");
    }
  }
}
```

2) WAP to check whether a person is eligible for IAS or not
eligibility is age b/w 22 and 35

Prog

```
public class Sample
{
 public static void main(String args[])
 {
  int a = 10;
  if(a>22 && a<35)
  {
   Sop(a+"eligible for IAS");
  }else
   Sop(a+" not eligible for IAS");
  }
 }
}
```

Output:- a not eligible for IAS

Prog:- WAP to check whether
A. 15 is divisible by 3 & 4
B. 12 is divisible by 3 & 4

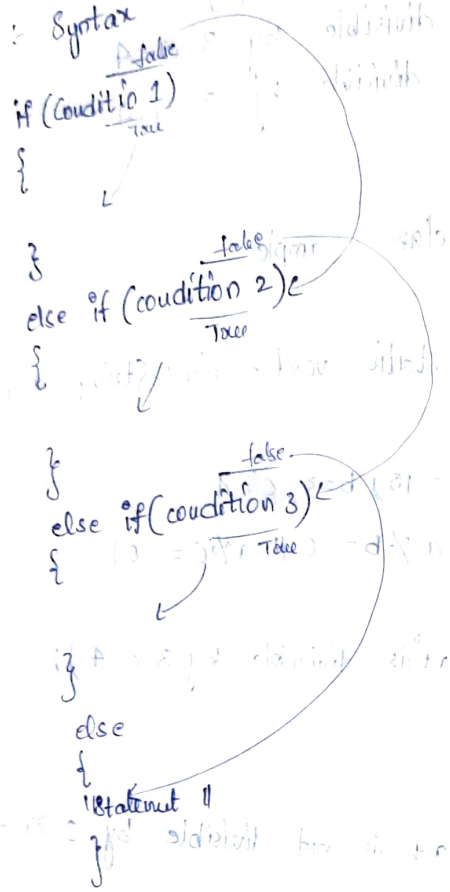
Prog:- (A)

```
public class Sample
{
 public static void main(String args[])
 {
  int a=15, b=3, c=4
  if(a%b==0 || a%c==0)
  {
   Sop(a+"is divisible by 3 & 4");
  }
  else
  {
   Sop(a+" is not divisible by 3 & 4");
  }
 }
}
```

(3)

```
int num = 15;
if(num%3==0 || num%4==0)
{
 Sop(" divisible);
}
else
{
 Sop(" not divisible");
```

TYPE 3 :- Syntax



Ex:- WAP to check greatest of three numbers
10 20 30

Prog:-

```
public class Greatnum
{
  public static void main(String args[]);
  {
    Int a=10, b=20, c=30;
    if (a>b && a>c)
    {
      Sop('a' + greatest");
    }
    else if (b>a && b<c)
    {
      Sop(b+ "greatest");
    }
    else if (c>a && c>b)
    {
      Sop(c+" is greatest");
    }
  }
}
```

Assignment

* WAP to check whether 10 & 20 are equal or not
* WAP to check whether 10, 20 & 30 are equal or not

Imp point

⇒ Integeral datatype ie:, byte, short, int & long can be represented in three ways

① defaultly for any integral data value is considered as decimal (base-10).

② If the number is starting with '0' it is considered as octal (base-8).

③ If the number is starting with 0x or 0X, number is considered as hexa decimal. hexadecimal (base-16).

Eg:

```
class Sample
{
  p s v m(String args[])
  {
    int x=10;
    int y=010;
    int z=0X10;
    Sop(x);
    Sop(y); // o/p → 8
    Sop(z); // o/p → 16
  }
}
```

Octal (base-8) 0-7
⇒ 8⁰×0 + 8¹×1 + 8²×0
⇒ 0+8+0 = 8

Hexadecimal (base-16) 0-9 A-F/a-f
⇒ 16¹×1 + 16⁰×0
⇒ 16

1) WAP to check 10 & 20 are equal or not

Prog:-

```
public class Equalnum
public static void main(String args[])
{
 int a=10, b=20;
 if(a==b)
 {
  Sop("a + "and" + "are equal");
 }
 else
  Sop("a + "and" + "b + "are not equal");
}
```

2) WAP to check 10, 20 & 30 equal

```
public class Equalnum
{
 public static void main(String args[])
 {
  int a=10, b=20, c=30;
  if(a==b && b==c)
  {
   Sop(a+b+c+"are equal");
  }
  else
  {
   Sop(a+b+c+" are not equal");
  }
 }
}
```

WAP to print name of the day by using below data

Day no	Day name
1	Monday
2	Tuesday
⋮	⋮
7	Sunday

Eg:

```
public class Days
{
public static void main(String args[])
{
 int dayno = 3;
 if (dayno == 1)
 {
  Sop("Monday");
 }
 else if (dayno == 2)
 {
  Sop("Tuesday");
 }
 else if (dayno == 3)
 {
  Sop("Wednesday");
 .
 .
 .
 else
 {
  Sop("Invalid dayno");
 }
```

Above program is correct but it is not preferrable because code is lengthy, unnecessary comparisons takes place and execution time also increases.

In such case where we have multiple conditions rather than using conditional statements we should use "switch case" statements.

SWITCH CASE :- (Syntax)

```
switch (expression)
{
case 1:
   // some statement //
   break;
case 2:
   // some statement //
   break;
   .
   .
   .
case n:
   // some statement //
   break;
default:
   // some statement //
   break;
}
```

Should same type as expression

Ex: Switch case

~~int num~~

```
int dayno = 3;
switch (number)
{
 case 1:
 Sop ("Monday");
    boreak;
 case 2:
 Sop ("Tuesday");
      boreak;
  case 3:
    Sop("Wednesday");
    boreak;
  case 4:
    Sop("Thursday");
    boreak;
   default:
      Sop("invalid no");
      boreak;
    }
```

Switch case Vs if else

=> Using switch case we can check multiple conditions at the same time with easy readability. We can also check with conditional statements but for a compiler it is complicated to check so many conditions because of readability.

=> This is one of the advantage of switch case over conditional statement

```
int day=3;
switch (day)
{
 case 1:
 case 2:
 case 3:
 case 4:
 case 5:
   Sop("Wednesday");
   break;
 case 6:
   Sop("Weekend-1");
   break;
 default:
   Sop("Invalid");
 break;
}
```

```
int day=3;
if(day==1 || day==2 || day==3
        || day==4 || day==5)
{
 Sop("Weekday");
}
else if(day==6)
{
Sop("first weekend");
}
 else if(day==7)
{
 Sop("second weekend");
}
else
{
 Sop("invalid day");
}
```

Loopholes for switch case

(A) Drawbacks :-

① Allowed data type :-

① Allowed datatype for cases values only byte, short, int, char datatypes are allowed. float, long, double & boolean are not allowed.

② till 1.6V, strings are not allowed but from 1.7V strings can be used as a case values.

② Omission of 'break' statement

=> Writing of 'break' keyword in cases is mandatory but writing of 'break' statement in default is not mandatory

=> Because once default is executed JVM will any how come out of switch case without break also but if we didn't write break after case statement, JVM will not come out of switch and continues execution till it finds break keyword.

```
int num=12;
switch (num)
{
 case 0:
   Sop("zero");
   break;
 case 1:
   Sop("one");
 case 2:
   Sop("Two");
   break;
 default:
   Sop("invalid");
 }
```

③ Duplicate cases

```
int num=3;
switch(num)
{
case 1:
    Sop("one");
    break;
case 2:
    Sop("Two");
    break;
case 3:
    Sop("Three");
    break;
case 3:
    Sop("four");
    break;
default:
    Sop("invalid");
    break;
}
```

Duplicate case values are not allowed

④ Case as variable

Case values as a variable are not allowed because case values should must be constant

Ex:

```
int num=3;
switch(num)
{
case 1:
    Sop("1");
    break;
case y:
    Sop("2");
    break;
}
```

→ not allowed [should be constant value]

* WAP to print java 5 times

Prog:

```
public class JAVA5
{
public static void main(String args[])
{
```

Iterative (or) Looping statements

If a part of a code is repeatedly executing then rather than writing it multiple times we can write it once and put it in a loop and we can execute how many times we want.

Because loop will run till the condition is false.

Syntax:

```
for(dec+ initialisation; condition, incre/decre)
{
  // statement //
}
```

Ex:

```
for(int i=1; i<=5; i++)
{
  Sop("java");
}
```

o/p :

JAVA
JAVA
JAVA
5 times

Variations of for loop :-

①
```
for(int i=1; i<=10; i++)
{
  sop( );
}
```
✓

②
```
for(int i=10; i>=1; i--)
{
  sop(..);
}
```
✓

③
```
for(int i=1; i>=1; i++)
{
  sop( ); // i
}
```
✓ inf

Here condition will never be false so loop runs infinite times

④
```
for(int i=1; i%2==0; i++)
{
  sop( );
}
```
✓

1%2 == 0
1 == 0

In first iteration only, condition is false so no o/p

o/p : BLANK

⑤
```
for(int i=1; i<=10; i+1)
{
  sop( );
}
```
→ Invalid Syntax ✗

WAP to print the num's from (i) 20 to 60 and (ii) 60 to 30

Prog :-

(i)
```
public class Increase
{
public static void main(String args[])
{
for(int i=20; i<=60; i++)
{
  sop(i);
}
```

o/p :- 20
21
22
23
:
60

(ii)
```
for(int i=60; i>=30; i--)
  sop(i);
```

(2) WAP to print (a) A to Z → 65 to 90
(b) a to z → 97 to 122

Prog:
```
public class Sample
{
public static void main (String args[])
{
 char i;
 for(i=65; i<=90; i++)
 {
  Sop(i);
 }
}
}
```

(or)
```
for (char ch='A';
     ch<=z;
     ch++)
 {
  Sop(ch);
 }
```

(3) WAP to print all even nos from 1 to 25

Prog:-
```
class sample
{
p s v m (String args[])
{
for(int i=1; i<=25; i++)
{
 if (i%2==0)
 {
  Sop (i);
 }
}
}
```

(4) WAP to print all nos divisible by 5 present b/w 1 to 35
```
public class Sample
{
 public static void main (String args[])
 {
  for (i=1; i<=35; i++)
  {
   if (i%5==0)
   {
    Sop(i);
   }
  }
 }
}
```

(5) WAP to print all nos from 1 to 100 excluding multiples of 7.
```
p s v m(    )
{
for (i=1; i<=100; i++)
{
 if (i%7!=0)
 {
  Sop(i);
 }
}
}
```

⑥ WAP to count all nos from 1 to 30 which are divisible by 5

Prog:

```
public class Sample
{
public static void main(String args[])
{
  int count=0;
 for(int i=1; i<=30; i++)
 {
  if(i%5==0)
   {
    count++;
   }
 }
  Sop("Count is:"+ count);
}
}
```

⑦ WAP to count all the nos from 20 to 35 which are divisible by 3

⑧ (i) Sum of all nos divisible by 5 b/w 1 to 30

(ii) Product

⑦ Prog:

```
public class sample
{
public static void main(String args[])
{
 int count=0;
 for(int i=20; i<=35; i++)
 {
  if(i%3==0)
  {
   count++;
  }
 }
  Sop("Count is:"+ count);
}
}
```

⑧ Prog

```
public class sample
{
public static void main(String args[])
{
 int sum=0;
 for(int i=1; i<=30; i++)
 {
  if(i%5==0)
```

```
        {
          sum = sum + i;
        }
      }
      Sop("sum value is:" + sum);
    }
  }
```

8(i)

```
int pdt = 1;
for (int i=1; i<=30; i++)
{
  if (i%5 == 0)
  {
    pdt = pdt * i;
  }
  Sop("Pdt value is:" + pdt);
}
```

9) WAP to print sum of first 10 natural no's

Prog

```
public class Sample
{
  public static void main(String[] args)
  {
    int sum = 0;
    for (int i=1; i<=10; i++)
    {
      sum = sum + i;
    }
    {
      Sop("Sum of the no's is": + sum);
    }
  }
}
```

10) WAP to print 5!

```
class Sample
{
  p s v m (String args[])
  {
    int fact=1; num=5;
    for (int i=1; i<=num; i++)
    {
      fact = fact * i;
    }
    Sop("& factorial is:" + fact);
  }
}
```

11) WAP to print the multiplication table of 5 in the given format

Prog:

```
public class Sample
{
 public static void main( )
 {
  int num=5;
  for (int i=1; i<=10; i++)
  {
   Sop( num + "*" + i + "=" + num*i)
  }
 }
```

(or)

```
int num=5, pdt=1;
for (int i=1; i<=10; i++)
{
  pdt= num*i;
 Sop( num + "*" + i + "=" + Pdt);
}
```

12)

Prog:

```
int num = 5;
for(int i=1; i<=10; i=i+2)
{
 Sop( num + "*" + i + "=" + num*i);
}
}
```

(or)

```
int num=3, Pdt=1
for(int i=1; i<=10; i++)
{ if (i%2!==0)
  { pdt= num*i;
  } Sop(num + "*" + i + "=" + Pdt);
}
```

13) WAP to check whether 15 is prime number or not

```
Class Sample
{
p s v m (String args[])
{
 int num = 15, count=0;
 for(int i=1; i<=num; i++)
 {
  if (num%i==0)
  {
   count ++; // count = count + i;
  }
 }
  if (count==2)
 {
  Sop("num is primenum");
 }
 else
 {
  Sop(num + "is not primenum");
 }
```

WAP to print all the nos from 1 to 100 which has digit 5

Prog:

```
public class Sample
{
psvm(String args[])
{
if(i%10==5 || i\10==5)
{
sop(i);
}
for(int i=1; i<=100; i++)
{
if(i%10==5 || i\10==5)
{
sop(i);
}
}
}
}
```

Executing a prgrm using command prompt :-

⇒ Install JDK by referring through document for checking whether JAVA is installed properly or not, go to command prompt and type command as JAVA - version.

⇒ Open notepad or notepad++ and develop the program.

⇒ Save the program as clsname.java and remember the location wherever we have saved our program.

⇒ Compilation: Once we go to command prompt defaulty the control will be C - drive.

⇒ For coming out of particular folder, enter the command as cd..

⇒ Command for entering into particular folder is cd.. foldername.

⇒ Once we enter into desired folder compile the program by entering command as javac programname.java.

⇒ If there is no compilation error classfile will generate successfully.

⇒ Execution: for executing the program enter the command as java clsname.

⇒ For clearing the screen enter the command as cls.

⇒ For changing the drive enter the command as drivename: Ex: E:

LOOP HOLE :

⇒ In for loop we can initialize more than one variable but it should be of same type and we should write datatype only once.

```
Ex:1  for(int i=1, j=1; i<=10; i++) // loop body //
      {
       //valid //
      }
      for(int i=1, int j=1, i<=10, i++)
      {
       // invalid //
      }
      for(int i=1, String s="A", i<=10; i++)
      {
       // invalid //
      }

Ex:2  int i,j;
      for(i=0, j=0, i<10; i++)
      {
       //valid //
      }
```

⇒ Giving flower braces in for loop is optional if we didn't write only one statement is considered as part of loop body, remaining all statement will get executed ouce cond of loop is false.

```
Ex:  {for(i=1; i<=10; i++)
        Sop("JAVA");
        Sop("SQL");
     }
```

O/P:-

```
JAVA
JAVA
:
:
10
SQL
```

⇒ If we put ";" in the end of for loop we will not get compile time error and will not get run time error also rather it will be pointed as an empty loop and ouce condition is false whatever we wrote inside the loop that will be executed only ouce.

```
Ex:  for(int i=1; i<=10; i++);
     {
       Sop("Java");
     }
```

O/P: Java

⇒ If we write didn't put condition will not get compile time error and also run time error rather JVM take it as true and loop will run infinite times

Ex:
```
for(int i=1; true; i++)
{
  Sop("Java");
}
```
O/p :- Java
:

Examples

①
```
int i, j, a=10, b=20
for(i=1, j=1; a<b; i++)
{
  Sop("Java"); // Infinite times
}
```

②
```
for(int i=1, j=1; i++, j++)
{
  Sop("Java"); // Infinite
}
```

③
```
for(int i=1; ; i++)
{
  Sop("Java"); // Infinite times
}
```

```
Sop("SQL"); // Compile time error because unreadable statement
```

④
```
for(int i=1, j=1; i<=10 && j<=10; i++, j++)
{
  Soplu("Java"); // Infinite
}
Sop("SQL"); // Out time
```

Pattern Programming :-

outer loop → No. of rows (lines)
Inner loop → No. of columns

```
for(int i=1; i<=3; i++)
{
  for(int j=1; j<=6; j++)
  {
    System.out.print("*"); → for printing *'s in same line
  }
  Soplu(); → out of inner loop
}
```
for breaking line & move cursor to next element

(9)

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

i	j
1	1
2	1, 2
3	1, 2, 3
4	1, 2, 3, 4
5	1, 2, 3, 4, 5

Prgm :-

```
public class sample
{
  public static void main (String args[])
  {
    for(int i=1; i<=5; i++)
    {
      for(int j=1; j<=5; j++)
      {
        if (i >= j)   (or)   if (i==j || i>=j)
        {
          Sop ("*");
        }
        else
        {
          Sop (" ");
        }
      }
      Soplu( );
    }
```

(10)

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

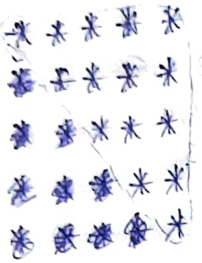
i	j
5	1, 2, 3, 4, 5
4	1, 2, 3, 4
3	1, 2, 3
2	1, 2
1	1

Prgm :-

```
class sample
{
p s v m ( )
{
  for(i=1; i<=5; i++)
  {
    for(int j=5; j>=1; j--)
    {
      if (i>=j)
      {
        S.O.print ("*");
      }
      else
      {
        Soprut (" ");
      }
    }
    Soplu( );
  }
```

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

(11)



```
Prgm:- for(int i=1; i<=5; i++)
       {
       for (int j=1; j<=5; j++)
       {
       if(i<=j)
       {
        Soprint("*");
       }
       else
       {
        soprint(" ");
       }
       }
        Soplu();
       }
```

i	j
1	
2	
3	
4	

(12)

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

Prgm:-

```
class Sample
{
p s v m(  )
{
 for(int i=4; i<=1; i++)
 {
  for(int j=1; j<=4; j++)
  {
   if(j>=i)
   {
    Sop("*□");
   }
   else
   {
    Sop(" ");
   }
  }
   Soplu();
 }
}
```

(13)

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

Prgm:-

```
for(int i=1; i<=5; i++)
 {
 for(int j=1; j<=5; j++)
 {
  if(i<=j)
  {
   Sop("*□");
  }
  else
  {
   Soprint(" ");
  }
 }
  Soplu();
}
```

(14)

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

Prgm:-

```
p   c   Sample
{
 p s v m(  )
 {
  for(int i=1; i<=3; i++)
  {
   for(int j=1; j<=3; j++)
   {
    Soprint(i);
   }
    Soplu();
  }
```

② 1 2 3
1 2 3
1 2 3

Prgm

```
p c Sample
{
p s v m( )
{
for(iut i=1; i<=3; i++)
{
for(iut j=1; j<=3; j++)
{ {
Sopriut(j);
}
Soplu( );
} } } }
```

③ 1 2 3
4 5 6
7 8 9

Prgm :-

```
iut k=1;
for(iut i=1; i<=3; i++)
{
for(iut j=1; j<=3; j++)
{
Sopriut(k);
k++;
}
Soplu( );
}
```

④ 1
2 2
3 3 3
4 4 4 4

Prgm

```
for(iut i=1; i<=4; i++)
{
for(iut j=1; j<=i; j++)
{
S.opriut(i);
}
S.oplu( );
}
}
```

⑤ 1
1 2
1 2 3
1 2 3 4

Prgm

```
for(iut i=1; i<=4; i++)
{
for(iut j=1; j<=i; j++)
{
Sopriut(j);
}
Soplu( );
}
}
```

⑥
```
1
2 3
4 5 6
7 8 9 10
```

Pgm:
```
int k=1;
for (int i=1; i<=4; i++)
{
for (int j=1; j<=4; j++)
{
  if(i>=j)
  {
    S.o.print (k);
    k++;
  }
}
 So.plu( );
}
```

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

Pgm:
```
int k=1;
for(int i=1; i<=4; i++)
{
for(int j=4; j>=1; j--)
{
  if (i>=j)
  {
    Sopt(i);
  }
  else
  {
    Sopriut(" ");
  }
 S.oplu( );
}
```

⑧
```
4 4 4 4
  3 3 3
    2 2
      1
```

Pgm:
```
for(int i=4; i>=1; i--)
{
for(int j=4; j>=1; j--)
{
  if(i>=j)
  {
    Sopt(i);
  }
  else
  {
    Sopt(" ");
  }
 Soplu( );
}
```

⑨
```
1
0 1
0 0 1
0 0 0 1
```

Pgm:
```
for(int i=1; i<=4; i++)
{
for (int j=1; j<=4; j++)
{
 if (i>=j && i==j)
 {
  Sopt(1);
 }
 else if (i>=j)
 {
  Sopt(0);
 }
 else
 {
  Sopt(" ");
 }
}
 Soplu( );
}
```

⑩
```
1
1 0 1
1 0 1
1 0 1 0
```

Pgm:
```
for(int i=1; i<=4; i++)
{
for(
```

① A A A
B B B
C C C

Program:-
```
for (char i='A'; i<='C'; i++)
{
 for (char j='A'; j<='C'; j++)
 {
  Sop(i);
 }
}
```

```
Char ch='A';
for(int i=1; i<=3; i++)
{
 for (int j=1; j<=3; j++)
 {
  Sop(ch);
 }
 ch++;
 Soplu();
}
```

(or)

```
}
```

② A B C
A B C
A B C

Prgm:-
```
char ch='A'
for(int i=1; i<=3; i++)
{
 for (int j=1; j<=3; j++)
 {
  Sop(j);
 }
}
```

③ A B C
D E F
G H I

Prgm:-
```
char ch='A'
for (int i=1; i<=3; i++)
{
 for (int j=1; j<=3; j++)
 {
  Sop(ch);
  ch++;
 }
 Soplu();
}
```

④ A
B B
C C C
D D D D

Prgm:-
```
for(char i='A'; i<='D'; i++)
{
 for(char j='A'; j<=i; j++)
 {
  Sop(i);
 }
 Soplu();
}
```

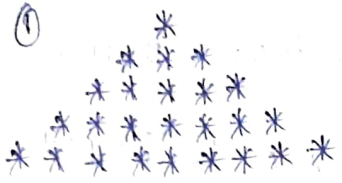
⑤ A
A B
A B C
A B C D

Prgm:-
```
for (char i='A'; i<='D'; i++)
{
 for (char j='D'; j<=i; j++)
 {
  Sop(j);
 }
 Soplu();
}
```

⑥ A
B C
D E F
G H I J

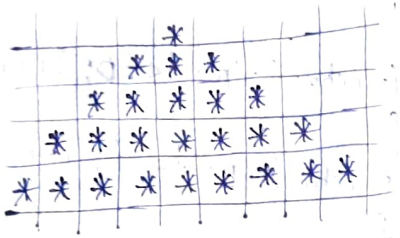
Prgm:-
```
char ch='A';
for(int i='A'; i<='D'; i++)
{
 for(int j='A'; j<=i; j++)
 {
  Sop(ch);
  ch++;
 }
 Soplu(ch);
}
```

(1)

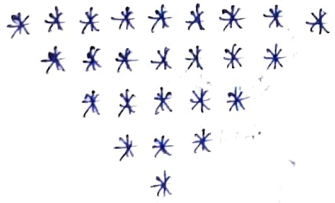


Pgm:-

```
class Pattern
{
p s v m (String args[])
{
int star=1;
    space=4;
for(int i=1; i<=5; i++)
{
for(int j=1; j<=space; j++)
{
So.print(" ");
}
for(int k=1; k<=star; k++)
{
S.O.print("*");
}
star = star+2;
space= space-1;
Soln( );
}}
}
```



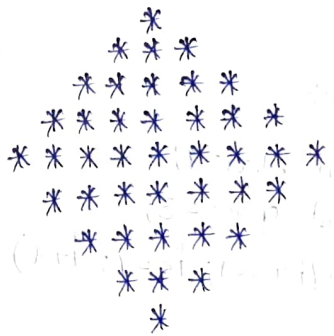
(2)



Pgm:-

```
public class Pattern
{
  p s v m( )
  {
    int star=9; int space=4;
  for(int i=1; i<=5; i++)
   {
  for(int j=1; j<=space; j++)
  {
  S.O.print(" ");
  }
  for(int k=1; k<=star; k++)
  {
  S.O.print("*");
  }
  star=star-2;
  space=space-1;
  Soln();
  }
}}
```

(3)



Pgm:-

```
public class Pattern
{
 p s v m( )
 {
  int star=1;
  int space=4;
for(int i=1; i<=9; i++)
{
 for(int j=1; j<=space; j++)
 {
  S.O.print(" ");
 }
 for(int k=1; k<=star; k++)
 {
   Sopoint("*");
 }
```

```
if (i<=4)
{
  star = star+2;
  space = space-1;
}
else
{
  star = star-2;
  space = space+1;
}
S.o.println();
}
}
}
```

④

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

```
}
if (i<=3)
{
  star = star-2;
  space = space+1;
}
else
{
  star = star+2;
  space = space-1;
}
Soplu();
}}}
```

Pgm:-

```
int star=7;
int space=3;
for (int i=1; i<=7; i++)
{
  Sop.println("*");
}
if(i<=3
for(j=1; j<= space; j++)
{
  sop(" ");
}
for(int k=1; k<=star; k++)
{
  Sop("*");
```

⑤

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

```
  star = star-1;
  space = space+1;
}
S.o println();
}
```

Pgm:-

```
int star=1, space=0;
for(int i=1; i<=7; i++)
{
  for(j=1; j<=space; j++)
  {
    S.o.println(" ");
  }
  for(int k=1; k<=star; k++)
  {
    S.o.println("*");
  }
  if (i<=3)
  {
    star=star+1;
    space=space-1;
  }
  else
  {
```

⑥

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

```
{
  star= star-1;
  space = space+1;
}
Soplu();
}}
}
```

Pgm:-

```
int star=1; space=3;
for(int i=1; i<=7; i++)
{
  for(j=1; j<=space; j++)
  {
    Sop(" ");
  }
  for(int k=1; k<=star; k++)
  {
    Sop("*");
  }
  if (i<=3)
  {
    star=star+1;
    space=space-1;
  }
  else
```

(1)

```
    * #
  * * # #
* * * # # #
  * * * # # # #
* * * * # # # # #
```

Pgm:-

```
int star=1;
int hash=1;
int space=4;
for(int i=1; i<=5; i++)
{
 for(int j=1; j<=space; j++)
 {
  Sop print(" ");
 }
 for(int k=1; k<=star; k++)
 {
  Sop print("* ");
 }
 for(int L=1; L<=hash; L++)
 {
  Sop("#");
 }
```

```
star = star+1;
hash = hash+1;
space = space-1;
Soplu( );
}
}
}
```

(2)

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

Pgm:-

```
for(int i=1; i<=5; i++)
{
 for(int j=1; j<=5; j++)
 {
  if(i==1 || j==1 || i==5 || j==5 || i==j)
  {
   Sop("*");
  }
  else
  {
   Sop(" ");
  }
 }
 Soplu( );
}
}
}
```

(3)

```
      A
    B   B
  C   C   C
D   D   D   D
```

Pgm:-

(4)

```
1 A A A A
1 2 B B B
1 2 3 C C
1 2 3 4 D
1 2 3 4 5
```

Pgm:- int

④

```
* * * * *
A * * * *
B C * * *
D E F * *
G H I J *
```

⑤

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

⑥

```
1
0 1
1 0 1
0 1 0 1
```

* No 4 is prime or not

Prgm:-

```
public class PN
{
 p s v m(  )
 {
  int num=4;
 boolean status = true;
 for( int i=2; i<num; i++)
{
  if (num%i==0) // Other than 1 and itself if any num divisible
  {                //
    status=false;
    break;
  }
 }
  if(status==true)
 {
   Sop(num+" is prime no");
 }
else
  {
    Sop(num+" is not a prime no");
  }
 }
}
```

* WHILE LOOP *

We go for while loop if when we want to perform some set of operations but we don't know how many iterations will happen.

Syntax :-

```
initialisation ;
while(condition)
{
  // loop body //
  increment/decrement ;
}
```

Ex:- Print java 10 times

```
int i=1;
while(i<=10)
{
  Sop("java");
  i++;
}
```

① WAP to reverse of num

Pgm:-

```
int num = 123;
int reverse = 0, mod result;
while (num > 0);
{
  modresult = num % 10;
     num = num/10;
  reverse = reverse * 10 + modresult;
}
  Sop("Reverse of num is:" + reverse);
}
}
```

② WAP to check paliudrome of number

Pgm:-

```
if (temp == reverse)
{
  Sop (temp + "is a paliudrome num");
}
else
{
  Sop ("Non paliudrome no");
}
}
}
```

③ WAP to check num is armstrong or not

def: 123 individual cube and sum should equals to 123.

Pgm:-

```
{
  int num = 123, temp = num;
  int reverse = 0, modresult;
  while (num > 0)
{
  modresult = num % 10;
     num = num/10;
reverse = reverse + (modresult * modresult *
                     modresult)
}
  Sop("Reverse of no is" + reverse);
  if (temp == reverse)
{
  Sop( temp + " is an armstrong number");
  else
{
  S.o.p("non armstrong num");
}
}
}
```

④ WAP to print fibnocci series

1 1 2 3 5 8 13

def: from 3rd num, every num is sum of prior to num's.

Pgm:

```
public class Fib
{
 p s v m(      )
 {
  int a=1, b=1, c;
  S.o.println(a+" "+b);
   for(int i=1; i<=10; i++)
  {
   c=a+b;
   S.o.println(" "+c);
   a=b;
   b=c;
  }
 }
}
```

⑤ Assignment :-

WAP to print all nos from 1 to 15

Pgm:-

```
{
 for(int i=1; i<=15; i++)
 {
  boolean status=true;
  for(int j=2; j<i; j++)
  {
   if(i%j==0)
   {
    status=false;
    breaks;
   }
  }
  if(status==true)
  {
   S.o.println(i+" is prime numbers");
  }
 }
}
```

"DO WHILE" LOOP :-

Syntax:-

```
do
{
 // loop body //
}
while(condition);
```

→ We go for do while loop when we want our loop should be executed atleast once.

→ In this loop first loop body will be executed then condition will be checked.

Ex:-

```
int i=1;
do
{
 Sop("Hello");
 i++;
}
while(i<=3);
```

NOTE:- (i) When we know no. of iterations we will use "for" loop.

(ii) When we dont know no. of iterations we will use "while" loop.

(iii) When we want our loop should be executed atleast once we will go for "do while" loop.

COMBINATIONAL OPERATORS :-

+=	⟶ a = a+b	⟷ a += b
-=	⟶ a = a-b	⟷ a -= b
*=	⟶ a = a*b	⟷ a *= b
/=	⟶ a = a/b	⟷ a /= b
%=	⟶ a = a%b	⟷ a %= b

Ex:

```
public class Sample
{
  p s v m( )
  {
    int a=10, b=20, c=30, d=40;
    a=a+b;
    S.o.plu(a);
    C+=b;
    S o plu (c);
  }
}
```

1) What is JAVA ?

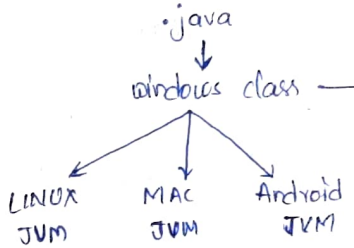
⇒ JAVA is a programming language which follows this principle.

WORA ⇒ Write One Run Anywhere

⇒ JAVA supports object oriented programming which is required to fulfil every project requirement.

⇒ JAVA is a platform independant language

⇒



⇒ Java program after into byte code can be run on any platform (OS).

But for running a program we need JVM of that particular platform.

So, JAVA is platform independant but JVM is platform dependant.

⇒ JAVA is a portable language – Along with java we can use any other language as a supporting language.

⇒ JAVA is a Robust language – Most of the important things will be done automatically.

Ex: Without calling garbage collector it will cleanup memory.

⇒ JAVA is a secured language.
ie., If anything happens JVM effects but it does not effect OS.

⇒ JAVA supports multi threading — We can execute multiple task parallely.

Difference b/w C-language and JAVA

C language	JAVA
① `#include<stdio.h>` `{` `void main()` `{` `printf("C");` `}` `}`	① `p c Sample` `{` `p s v m( )` `{` `Sop( );` `}` `}`
② Contains header files ↓ Predefine funct	② Contains Packages ↓ class — methods
③ OS takes responsibility	③ JVM " "
④ Depends on O.S	④ Doesn't depends on O.S
⑤ Compiled language	⑤ Compiled + Execution

* Every program consist of data and operations.
⇒ Data is nothing but variables
⇒ Operations are nothing but methods.
* In java we have predefine methods as well as we can create our own methods.

Syntax for creating a method is:

(Access modifier) (Non Access modifier) (Return type) (method name (args/no args)) { — Method header
method body/implementation
} — Method signature

* Methods are used to perform some operations.
Ex:- Perform airthematic operations
To print of area of Δ, ○, ▭
To print employ information etc

Ex:-
```
public static void airth operations( )
{
    Sop(10+20);
    Sop(10-20);
}
```

Ex:-

```
class Info
{
    public static void main(String args[])
    {
        S.oplu("main method");
        rani(); // call to rani()
        Sop("main ends");
    }
    public static void rani()
    {
        So.plu("Will rani answer my call");
    }
}
```

Predefine method

User define

O/p :-
main method
Will rani answer my call
main ends

* In a program along with main method we can have multiple user define methods
* Whenever we create any user define method it is our responsibility to call that method
* Without calling that method will not be executed.
* Placement of main method is not important because wherever we write main method execution will always starts from there

* We can call a method multiple times

Ex:-

```
class Info
{
    public static void run()
    {
        walk(); // call to walk()
        So.plu("Someone - - ");
    }
    public static void walk()
    {
        sleep(); // call to sleep()
        Soplu("Someone is walking");
    }
    public static void sleep()
    {
        Soplu("Most beautiful");
    }
    public static void main(String args[])
    {
        Soplu("Main starts");
        run(); // call to run()
    }
}
```

O/p :-
Main starts
Sleeping Most beautiful
Someone is walking
Someone is running

* From the above program we can observe that we can not only call a main method but we can also call it from user defined method.

* We can call any no.of userdefined methods but we cannot call main method because JVM is responsible to call main method.

Method with arguments :-

Some methods requires additional inputs for that we can define a method with arguments

Syntax:- Access mod | Non access mod | Return type | method name (datatype var1, datatype var2, - - etc

* While calling such method we have to pass the values for those argument type.

Ex:- run( ) ⟶ call to run without args
run(1000) ⟶ call to run with integer args
run("java", 'A') ⟶ call to run with args as string and char

Ex pgm:-

```
class Info
{
 p s v run(int time, string s)
 {
  S.oplu(s+" is running from "+time+" hrs");
 }
 p s v m(string args[])
 {
  S.oplu("Main starts");
  run(2, "john"); // call to run() with args as integer and string
 }
}
```

① WAP to create add(int i, int j)
Sub(int i, int j)
multi(int i, int j)
div(int i, int j)
mod(int i, int j)

② WAP to create facebook (String username, int Pwd)
Instagram(String UN, int Pwd)
Snapchat(String UN, String e-id)

① Pgm:-

```
public class Operations
{
 public static void main(String args[])
 {
```

26/02/2020

Method with return type :-

Whenever we want our method to be specific to a particular return type we can use a method with specific return type.

Syntax : Access mod | Non access mod | Any primitive/ Non primitive data type | MN (with & without arg)

Ex:
```
Public static double multiply( )
{
    return (double value);
}
public static int add( )
{
    return (int value);
}
```

⇒ return keyword is used to exit from the method by taking the value.
⇒ return keyword will not print the value.
⇒ return keyword will not print the data if we want the data to be printed we have to call a method with printing statement.

Ex:-
```
public class MethodOv1( )
{
  p s v m(      )
  {
    Soplu (add(10,20));
  }
  public static int add (int i, int j)
  {
    int k=i+j;
    return k;
  }
```

⇒ In a method we can write only one return statement. And return statement must be the last statement of a method.

⇒ If we write more than one return statm or any statement after return statement we will get compile time error.

Ex:-
```
public class Methodov1
{
  p s v m(      )
  {
    Soplu (add (10,20));
  }
  public static int add (int i, int j)
  {
    int k=i+j;
    return k;
  X return k;
  X Soplu ("end of add");
```

WAP to perform operations (i) multiply (123, 27)
(ii) division (23,4)
(iii) mod (int i, int j)
44, 13

(i) Prog:-
```
public class Methodov1
{
  p s v m(      )
  {
    Soplu(multiply (123,27));
  }
  public static long multiply (int i, int j)
  {
    int k=i*j;
    return k;
  }
}
```
O/p:- 3321

(ii) Prgm:-
```
public class methodov1
{
  p s v m(String[] args)
  {
    Soplu ( division (23,4))
  }
  public static float division (int i, int j)
  {
    int k= i/j;
    return k;
  }
}
```
O/p :- 5

WAP to find area of a (i) triangle
(ii) Rectangle
(iii) Circle

Pgm (i):-

```
public class Triangle
{
public static void main ( )
{
  Soplu(triangle(10,20));
  Soplu (rectangle(6,8));
  Soplu(circle(5.0f));
}
public static float triangle (int b, int h)
{
  float k=0.5f * b* h
  return k;
}
public static int rectangle (int l, int b)
{
  int k= l*b;
  return k;
}
public static float circle (float radius)
{
  float pie = 3.14f
  float k = pie * radius * radius
  return k;
}
}
```

O/P:- 100
48
785

METHOD OVERLOADING :-

The process of developing multiple methods with the same name but different argument list is called as method overloading.

Rules for defining argument list

(1) Between the methods no. of arguments should be different

Ex:- marriage ( )
marriage (int number)

(2) B/w the methods type of argument should be different (type refers to data type)

Ex:- marriage (String type, ~~int num~~)
marriage (int number)

(3) B/w the methods sequence or position or place of arguments should be different.

Ex:- marriage (String type, int num)
marriage (int num, string type)

=) We go for method overloading, when we want to perform same task or one task in multiple ways

Ex:- Hyd → Goa
Car
bike
bus
-Train

* B/w two methods only one scale should applicable.

WAP to demonstrate method overloading

Prgm:-

```
public class Sample
{
public static void main( )
{
  System.out.println(add(5,2));
}
public static void add(int i, int j)
{
  Soptu int k = i+j;
        return k;
}
public static
```

Type Promotion in Method overloading :-

In method overloading when JVM did not find expected type of argument it will make a search for possible type promotion, As per the flow chart.

class sample

byte → short → int → long → float → double

char → int

Ex:-

```
class Sample
{
  p s v run( )
  {
    Sop("def run");
  }
  p s v run(float fl)
  {
    Sop("char run");
  }
  p s v main(String args[])
  {
    run( );
    run(A);
  }
```

In the above program JVM is searching for run method whose arg type is char.

If it finds char type argument program will execute happily, if it didn't find it will search for next possible type.

For a char, next possible type is int, long, long and double.

If there is no next possible type we will get compile time error.

```
public class Sample12
{
  p s v m( )
  {
    run( );
    run('A'); // CTE because there is no such method
  }
  p s v run(byte f)
  {
    Soplu(f);
  }
  p s v run()
  {
    Soplu("def run");
  }
}
```

* Can we overload main method or not?

Yes, we can overload because JVM will make a search for main method which has the args as command line args i.e; String args[]

If we write any other method even though the name is main JVM will not confuse b/w user define and predefine main method.

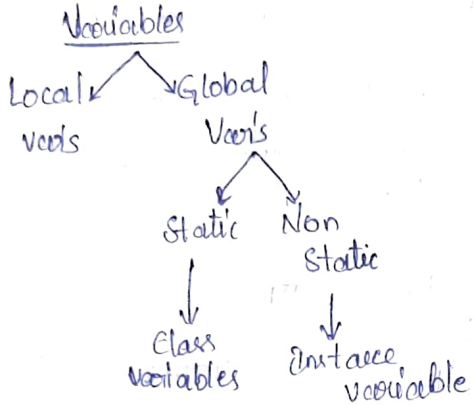
Ex:-

```
class Sample12
{
  p s v m(String args[])
  {
    main( );
    main(100);
  }
  p s v m(int i)
  {
    Soplu();
  }
  p s v m()
  {
    Soplu("def run");
  }
}
```

In practical it is not recomended to overload main method.

Types of variables:

Depending on decleration variables are divided into two types, they are



In an interview we have to answer this question as

=> Variables are of three types, ~~four~~ one

1) local variable
2) Static variable
3) Non static " (or)
4) Instance "

① Local variable :-

It is a variable which is decclared inside the method within the scope of a class. (or)

These are the variables which are decclared inside the method, with the methods integer, inside the constructor and with the constructor is called as

=> Local variables are accessable only within the scope of declared method. ie., local variables are not accessable outside the method.

Ex:-

```
public class Main
{
  p s v m(String args[])
  {
    int i=100;
    add();
    Soplu(i);
  }
  p s v add()
  {
    int a=10, b=20, c
    c=a+b;
    Soplu("Add result is"+c);
  }
}
```

O/P :- 1. Add result is 30
2. 100

=> There are only two accessable modifiers allowed for local variables ie., default and final

Global variables (&) Static variables

these are the variables which are declared outside the method within the scope of a class.

Global variables are accessable everywhere within the scope of a class.

These are two types, static and non-static

If a global variable contains static keyword it is static variable.

Ex:- Static int i=100
int j = 200

If a global variable doesn't contain static keyword it is non static variable.

Ex:-
```
public class Main
{
  p s v m(String args[])
  {
   int i= 100;
   add( );
   Soplu(i);
  }
  Static int i=100; float i = 100.0f
  public static void add( )
  {
   int j=200;
  float int c;
     c=i+j ;
   Soplu(c);
  }
}
```

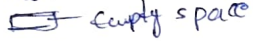
o/p:- 100
300.0

⇒ Initialisation of a global variable is not mandatory. If we didn't initialise JVM will take default values depending on data types and they are

byte, short, int, long } 0
float, double } 0.0
boolean } false
String } Null
char } Empty space

Ex:-
```
public class Sample
{
  static int i;
  static byte b;
  static short s;
  static long l;
  static boolean bl;
  static String Sl;
  static char ch;
  static float f;
  static double d;
```

```
public static void main (String args[])
{
    Sop(i);
    Sop(b);
        .
        .
        .
    Sop(d);
}
}
```

o/p:-
0
0
0
0
false
null
 empty space
0.0
0.0

⇒ We can use any access modifier for global variable like public, private, protected & default.

⇒ If local variable name and global variable names are same then first priority will be given to local variable only.

Ex:-
```
public class Sample
{
    static int i = 100;
    public static void main(String[] arg
    int i = 200;
    Sop(i);
}
}
```

o/p:- 200

1) Create 3 static variables

```
String s;
int i=100;
int j=200;
```

Prgm:-
```
public class Sample
{
    static String s = "SQL";
    static int i = 100;
    static int j = 200;
    public static void main(Stri
    {
        Sop(s);
        Sop(i);
        Sop(j);
    }
}
```

o/p:-
SQL
100
200

2) Create a static method as sum( )
create 2 local variables
one with same and another with diff
print all variables

Prgm:-

```
public class Sample
{
  ps v m( String args[])
```

3) Create main method
Create 2 local variables
print local and global variables values
to call to sum( )

Combine above 3 methods

Prgm:-

```
class Sample
{
  static String s;
  static int i=100;
  static int j=200;
  p s v sum( )
  {
    int i= 250;
    double d = 22.205;
    Sop(i);
    Sop(d);
    Sop(s);
    Sop(i);
  } Sop(j);
  p s v main(String args[])
  {
    int i= 700;
    float f= 12.05f;
    Sop(i);
    Sop(f);
    Sop(s);
    Sop(i);
    Sop(j);
    sum( );
  }
}
```

o/p:-

```
700
12.05
null
700
200
250
22.205
```

```
null
250
200
```

Diff b/w local and global variables

LOCAL VARIABLE	GLOBAL VARIABLE
① It is declared inside the method and within the scope of a class.	① It is declared outside the method and within the scope of a class
② Not Accessable everywhere only within the scope of declared method.	② Accessable everywhere within the scope of a class.
③ Local variables contains non-static variables.	③ Global variables contains static variables.
④ Non-static should not be written, it takes defaultly.	④ Static keyword should be written

Accessing members of a class :-

03 Mar 2020

```
public class Sample03
{
  static int i=100;
  int j=200;
  public s v m(String[] args)
  {
    Sample03 s1= new Sample03();
    s1.j=300;
    s1.i= 400;
    So.plu(s1.j); // 300
    So.plu(i); // 400
    Sample03 s2 = new Sample03();
    So.plu(s2.j); // 200
    So.plu(i); // 400
  }
}
```

Multiple ways to access static members

There are three ways

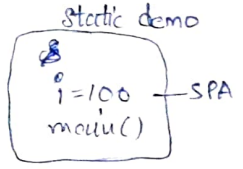
1) Directly
2) Through clsname
3) Through object

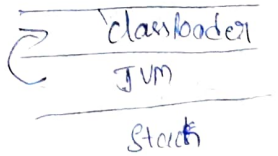
=> Static members can be accessible directly because they are single copy.

=> Static members can be accessable through clsname because clsname and static pool area names are same.

=> Also access through object because there is a one way connection from object to static pool area.

```
main()
Sop(i);
Sop(staticdemo.i)
staticdemo d1= new staticdemo();
Sop(d1.i);
```





Diff b/w static and non static

STATIC	NON STATIC
1) Static means single copy because it will get loaded only one in SPA.	1) This means multiple copies and it will get loaded only when we create an object
2) Static can be accessable in three ways. (i) Directly (ii) Through classname (iii) Through object	2) Accessable can be access through object
3) Static members are present in SPA.	3) Present in object.
4) Class loader is responsible for to load all static members	4) New keyword is responsible to load all non static members.
5) Static variable have single memory space.	5) Non static variable has seperate memory with every object.
6) We go for static if the data is fixed.	6) We go for this if the data is varying.
7) Static methods cannot be inherited.	7) These can be inherited
8) These are cannot be overwritten	8) Can be overwritten

Ex: Static string college = "GEC"; String campus = "XYZ"

Java Source File structure:-

```
class A
{

}
class B
{

}
class C
{

}
```

Save it as Sample.java

Comand: javc sample.java
→ A class
B class
C class
→ java Sample

⇒ Here program will get compiled since there is no mistake in the syntax.

⇒ But execution will not happen since there is no main method

Ex:
```
class A
}
class B
}
class C
{
    main()
}
```

→ Here it should be saved as C.java since main method is given.

Ex:3

```
class A
{
  main()
  {
  }
}
class B
{
  main()
  {
  }
}
class C
{
  main()
  {
  }
}
```

⇒ Here we can save as A.java / B.java / C.java / sample.java

⇒ Javac Sample.java

A.class
B.class
C.class

⇒ Now execute → java Sample (suppose)

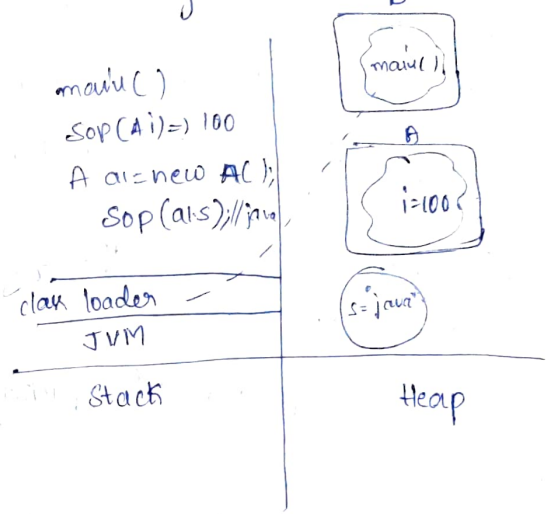
Error :- Could not find main in sample class

Accessing members of one class in another class :

```
class A
{
  Static int i=100;
  String s="java";
  → members of a class
}
```

```
class B
{
  P S V m(String args[])
  {
    Sop(A.i);
    // B b1=new B();  → loads all non static members of B class
    A a1=new A();
    Sop(a1.s);   s="java"
  }
}
```

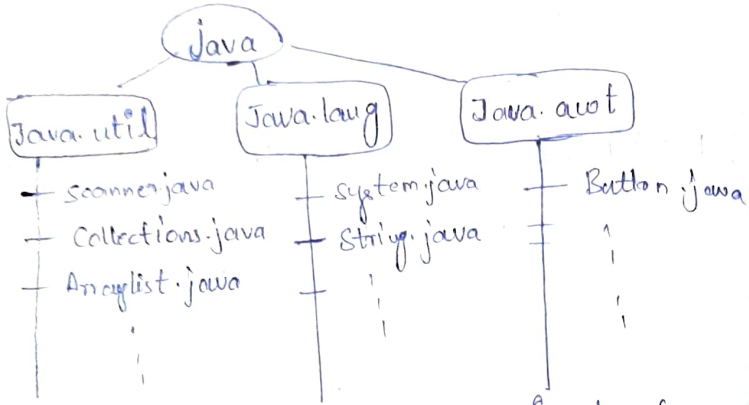
Save as B.java



⇒ In the above program first JVM will make a call to class loader and it loads all the static members of class B.

⇒ Once JVM starts execution first time whenever it encounters any new class it will make a call to class loader and class loader loads all the static members of that class and execution continues till the end of main method

Scanner class :



If we want to take any input from the user we have to make use of scanner class

Scanner class is present in java.util package

Package :

It is like a folder where all the common classes kept at one place.

Java contains several predefined packages like (above chart).

Class is present in one package, can be used in our own program by using import keyword.

It is a keyword which allows us to access data of one class in our own class.

Syntax for using import keyword is

1) Import Package.* ;
   name

   Ex: Import Java.util.* ;

2) Import Packagename.clsname ;

   Ex:- Import java.util.Scanner ;

⇒ Java contains one default package called as java.lang. ie., if we want to use classes of that package. we can use directly without import keyword.

Creating object of scanner class :

Scanner sc = new scanner (System.in)

⇒ System.in indicates we are taking an input through system.

⇒ In the above syntax with the new keyword one object gets created and it reads all the non-static methods of scanner class.

scanner sc = new scanner (System.in)

nextInt()
nextFloat()
nextDouble()
next.Byte()
next Boolean()
nextlong()
nextShort()
next()
nextLine()

Methods for taking char inputs from user is

① next Line().charAt(0) (or) ② next().charAt(0)

Nextint method:
It is used to take integer inputs from the user.

Nextfloat: Used to take float inputs " ".

NextDouble:- Used to take double inputs.

Next Byte:- Used to take bygte inputs

Next Boolean: Used to take booleau input

Next():- Used to take storiug iuput without any spaces.

NextLiue method:- Used to take striug iupeut with spaces.

Ex pergm:-

```
import java.util.*;

public class Demonstration
{
    p s v m(striug args[])
    {
        scanner sc = new scanner(System.in);
        Soplu("tuter integer iuput: ");
        int i=sc.nextint();
        Soplu("Enter float input: ");
        float f=sc.nextFloat();
        Soplu("Enter striug input: ");
        String s=sc.nextLiue();
        Soplu(i+" "+f+" "+s);
    }
}
```

① WAP to perform auithematic operatious usiug scanner class

Pgm:-

```
import java.util.*;

public class Operations
{
    p s v m(striug args[])
    {
```

Multiple ways to initialise non static variable:

First approach:-

```
class student
{
  String studentname; // non static instance var
  p s v m(String args[])
  {
    student s1= new student();
    Sop(s1.studentname);
  }
}
```

If we did not initialise any value for a non static variable, we will not get compile time error another JVM will take default values for every object. Which is not a recommandable approach.

Second approach:

```
class student
{
  String studentname = "John";
  p s v m(String args[])
  {
    student s1 = new student();
    Sop(s1.studentname);
  }
}
```

=> If we initialise non static variable while decleration for every object JVM will take same value.

This is also not a recommandable approach.

Third approach:-

```
class student
{
  String studentname();
  p s v m(str..)
  {
    student s1= new student();
    s1.studentname= "John";
    Sop(s1.studentname);
  }
}
```

=> If we go by this approach there might be a chance of programmer to forget about initialisation.

=> In such case again JVM will take default values.

=> Hence it is also not recommandable approach.

=> Therefore for proper initialisation of a non-static variable we go for constructor.

Constructors :-

Constructor is a special type of method which gets executed whenever we created an object.

NOTE:- It is called as special type of method because syntax is similar like a method.

Syntax:

```
Access modifier  Constructor name (args/no args)   <-- classname
{
    // constructor body //
}
```

Rules for defining a constructor :-

=> Rule 1:- Constructor allows access modifiers like public, private, protected and default.

=> Rule 2:- Constructor does not allows non access modifier like static, non-static, final and abstract.

=> Rule 3:- Constructor does not allows return type. not even void.

=> Rule 4:- Constructor name must be same as that of classname.

=> ~~Rules~~

Types of constructors :-

Two types

1. No argument constructor
2. Parameterised constructor

But we actually have these types of construct

1) Default constructor
2) No argument constructor
3) Parameterized constructor

ex:-

```
class student
{
    String studentname;
    /*
    Student()
    {
    // Empty implementation //
    }
    */
    p s v m (String args[])
    {
        Student s1 = new student();
        Sop(s1.studentname)
        Student s2 = new student();
        Sop(s2.studentname);
    }
}
```

default constructor created by JVM

Student name = null

null

null

Default constructor :-

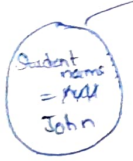
If a constructor does not contains any arg it is called as default constructor and it will be added by JVM but not programmer.

No argument constructor :-

If constructor does not contains any arg's it is called as no arg constructor and it will be added by programmer.

Ex:- ①

```
class student
{
  String studentname;
  public student( )
  {
    studentname = "John";
  }
  p s v m(String args[])
  {
    Student S1 = new student( );
    Sop(s1.studentname);
    Student S2 = new student( );
    Sop(S2.studentname);
  }
}
```



Ex:- ②

```
public class Dog
{
  String Dogname;
  String colour;
  public Dog( )
  {
    Dogname = "Tommy";
    Dog colour = "Pink";
  }
  public static void main(Str...)
  {
    Dog d1 = new Dog( );
    S o p ("Name is" + d1.Dogname + "Its colour is" + d1.colour);
    Dog d2 = new Dog( );
    Sop("Name is" + d2.Dogname .... + d2.colour);
  }
}
```

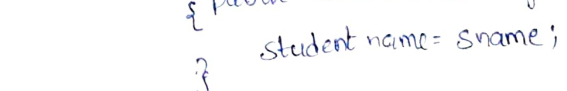
Parameterized constructor :-

If a constructor contains any arguments it is called as parameterized constructor.

NOTE:- Constructor arguments are always be local variables

Ex:-

```
class student
{
  String studentname;
  public student(String Sname)
  {
    student name = Sname;
  }
```



```
p s v m(Stri  )
{
    Student s1 = new Student("John");
    S o p (s1.studentname);
    Student s2 = new Student("Rohan");
    S o p (s2.studentname);
}
}
```

Student name = John

Student name = Rohan

Ex:- 3

```
class Dog
{
    String Dogname;
    p s v m(String --- )
{
    public Dog(String Dname)
    {
        Dogname = Dname;
    }
    p s v m(String args[])
{
    Dog d1 = new Dog("Jimmy");
    S o p (d1.Dogname);
    Dog d2 = new Dog("Tommy");
    S o p (d2.Dogname);
}
}
```

Tommy

Jimmy

Ex:- 4 :-

```
public class Dog
{
    String Dogname;
    String Dogcolour;
    public Dog(String Dname, Dcolour)
{
    Dogname = Dname;
    Dogcolour = Dcolour;
}
p s v m(Str---)
{
    Dog d1 = new Dog("Tommy", "Pink");
    S o p ("Dog colour is" + d1.dogcolour +
                "Dog name is" + d1.dogname);
}
}
```

Pgm:-

```
public class Book
{
    String bookname;
    int bookprice;
    int bookpages;
    String bookauthor;
```

```
public Book (String bname,int bprice,int bpages,
                                    String bauthor)
{
    bookname = bname;
    bookprice = bprice;
    book pages = bpages;
    book authoor = bauthoor;
}
p s v m (Str - - - )
{
  Book bl = new Book ("JAVA", 500, 300, "ITSME")
    S o p ("bookname" + bprice + "Rs" + bpages
                                    + "ITSME")
    ("bookname is" + bl.bookname + "bookprice is" +
        bl.bookprice + "bookpages is" + bl.bookpages +
            "book authoor is" + bl. bookauthor);
  Book b2 = new Book ("SQL", 300, 200, "SOMEONE")
Sop("bookname is" + bl.bookname + "bookprice is" + bl.book
    price + "book pages are" + bl.bookpages + "book auth
    is" + bl.bookauthor);
}
}
```

O/p:-

* WAP to print sum of squares of even num's from 1 2 3 4 8 2

Pgm:

```
public class Sum
{
 p  s  v  m (String args[])
{
  int num = 1 2 3 4 8 2 ;
  int digit, sum = 0;
  while (num>0)
  {
    digit = num % 10 ;
    num = num\10;
  if ( digit %2 == 0)
{
  sum = sum + (digit * digit);
}
}
  Soplu ("sum of squares of even nums are : + sum);
}
```

10) 1 2 3 4 8 2 (1 2 3 4 8
1 2 3 4 8
2

sum = 0
= 0 + (2*2)
= 2^

digit = 2,8
num = 2|10; 8/10 = 2+(8×8)
sum = 0+(2*2) = 2^+8^+(4×4)
= 0+2^
= 2^+(8×8) = 2^+8^+4^+(2×2)
= 2^+8^+4^+2^

* WAP for leap year

Parameterized constructor:

```
class Book
{
String bname;
int bprice;
public book(String bname, int bprice)
{
   bname = bname;  -> null
   bprice = bprice;  -> 0
}
 p s v m (String args[])
{
 Book b1 = new Book ("java", 500)
     b1.display( );
  Book b2 = new Book ("SQL", 300);
       b2. display( )
     }
    public void display( )
    {
      S o p (bname);  -> null
      S o p (bprice);  -> 0
    }
}
```

=> If we keep non static variable and coustructor arg names has same while initializing JVM could not differentiate b/w them without initializing it will come out of constructor and we know that a non-static variable is not initialize JVM will take default values.

Ex:- Above program

* Therefore to overcome this problem of same names java has given "this Keyword".

"this" KEYWORD :-

"This" is the keyword which indicates current object under execution.

It can be used inside a method and inside a constructor.

Ex:-

```
class Book
{
 String bname;
  int bprice;
public Book ( String bname, bprice)
{
 this.bname = bname; || b1.bname = bname
 this. bprice = bprice; || b1.bprice = bprice
}
  p s v m (String args[])
{
   Book b1 = new Book ("java", 500);
     b1. display( );
   Book b2 = new Book ("SQL", 300)
      b2. display ( );
}
 public void display
{
   S o p (bname);
   S o p (bprice);
}
}
```

Ex:-
```
public class Dog
{
    String dname;
    String dcolour;
public Dog (String dname,
            String dcolour)
{
  This.dname = dname;
} This.dcolour = dcolour;
  public void display()
  {
    Sop("Name is" + dname + "is colour"
                  + dcolour);
  }
  P S V m (String args[])
  {
    Dog d1 = new Dog ("Tommy", "Black");
       d1.display();
    Dog d2 = new Dog ("puppy", "Black");
       d2.display();
    Dog d3 = new Dog ("Sony", "white");
       d3.display();
  }
```

CONSTRUCTOR OVERLOADING :-

The process of developing multiple constructors with the same name but different argument list is called as constructor overloading.

Rules for defining argument list :-

* B/w the constructors no. of arguments should be differ.
* B/w the constructors type of arguments should be diffa.
* B/w the constructors position (or) sequence of argument should be different.

Ex:-
```
public class Add
{
  public Add()
  {
    S o p("In a default add
           constructor");
```

Program for constructor overloading :-

Prog :-
```
public class Sample
{
```

Chaining Constructors :-

The process of calling one constructor from another constructor is called as constructor chaining.

This can be achieved in two ways

① Call to this()

② Call to super()

① this() :- The process of calling one constructor from another constructor of same class.

Ex:

```
public class Add
{
public Add()
{
  this(100);
 System.out.println("default add constructor");
}
public Add(int i)
{
  this('A');
  S o p("Integer constructor"+ i);
}
public Add(char ch)
{
  S.o.p("char add constructor"+ ch);
}
public static void main(String args[])
{
  Add a1 = new add();
}
}
```

⇒ We cannot call a constructor from the same constructor

Ex:

```
public Add()
{
  this();
  S o p ("something");
}
```

⇒ this() must be the first statement in a constructor. ie, we cant write this() statement in any other line apart from first line we will get compile time error.

Ex:-

```
public Add()
{
  S o p ("something");
 this(); // CTE, because this() must be first statement
}
```

METHOD	CONSTRUCTOR
① Methods are used to perform some specific operations.	① These are used to initialise
② To execute a method we have to call it directly & through object.	② To execute this we have to create an object.
③ It can be static & non static.	③ It cannot be static & non static.
④ It can have any return type.	④ It doesn't have any return type not even void.

⑤ Method can be anything. ⑤ Same as that of clsname.

⑥ Methods can be inherited. ⑥ Cannot be inherited.

Class and object :

Class : A class is a blueprint or template of an app.

Object : An object is mirror image of a class.

⇒ Class is a logical entity, because it represents logic of an application.

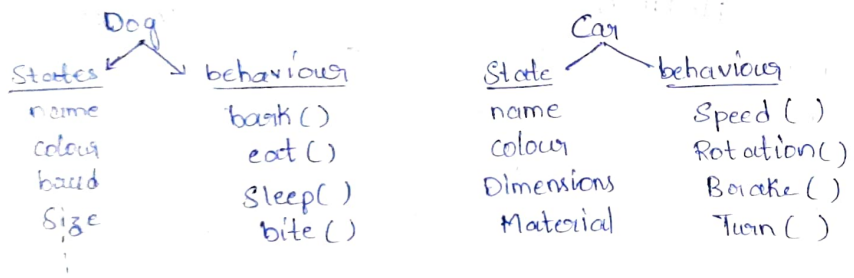
⇒ Object is a physical entity, because it contains some memory for a non static variable.

⇒ Every object represents two things, states and behavior

* State represents what object holds
* Behaviour represents what object does.

⇒ States are nothing but variables and behaviours are nothing but method.

Ex:

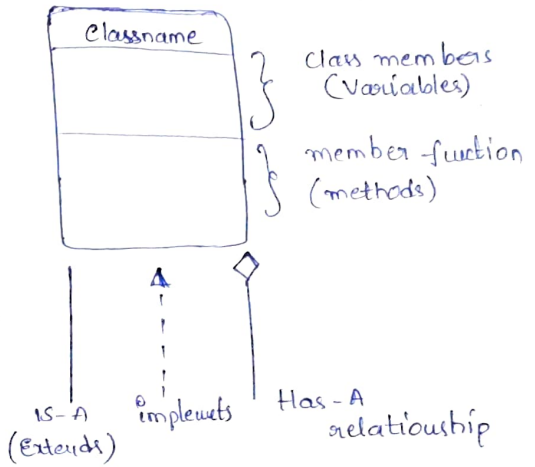


UML / Class diagram :

UML → Unified Modeling Language

It is the pictorial representation of an application.

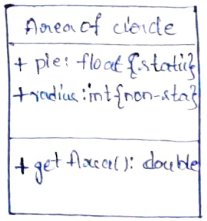
⇒ In real time developer will document the application by using UML / class diagrams.



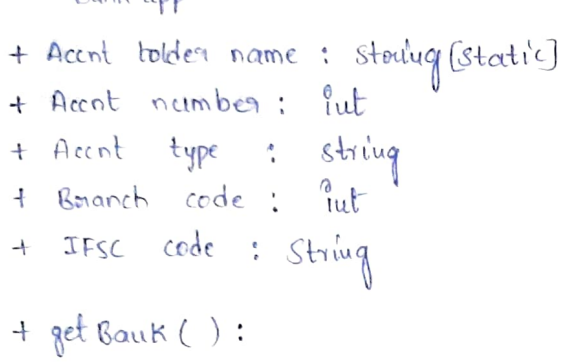
Access modifiers

+ → public
− → private
→ protected
~ → package (default)

Class dia for area of Circle :



Create a class dia for bank application :-



* OOPS *

11 Mar, 2022

It provides the concepts and methodologies to fullfill every project requirements.

It consists of four pillars

1. Inheritance
2. Polymorphism
3. Abstraction
4. Encapsulation

1. Inheritance :-

It is the first pillar of oops concept

* One class acquiring (accessing) the properties of another class is called as inheritance.
* The class who is acquiring the properties is called as child class (or) derived class (or) ~~super~~ child class.
* The class whose properties are acquired is called as parent class or base class (or) super class. Sub
* Super class have its own properties.
  Sub class have its own properties and also super class.

Extends :-

It is a keyword which indicates, we are deriving a new class from an existing class.