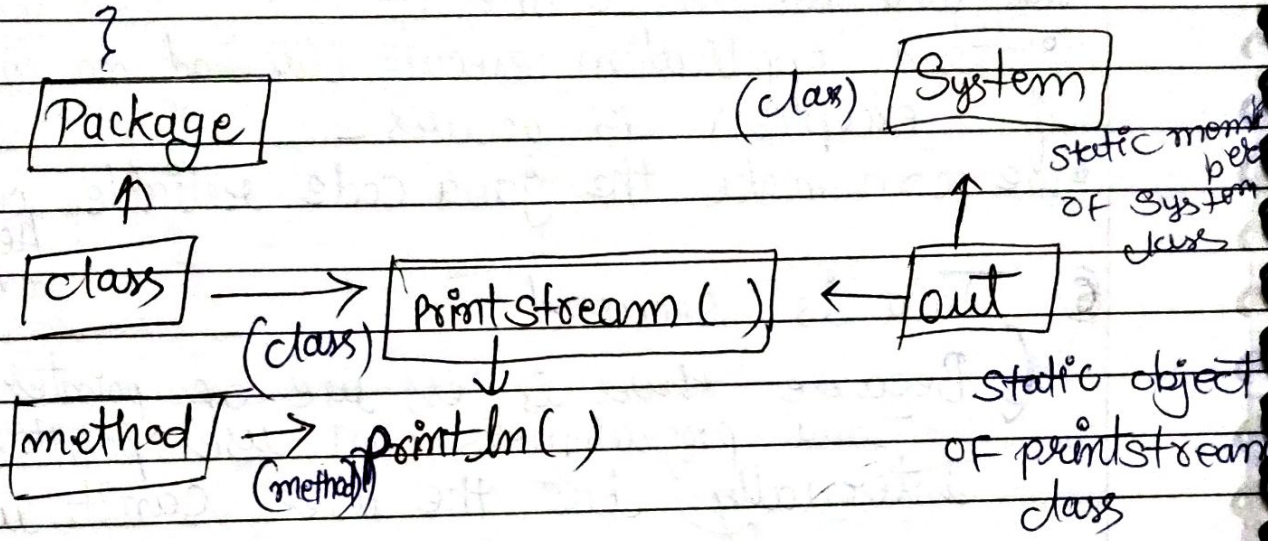
First Java program :-

class Name

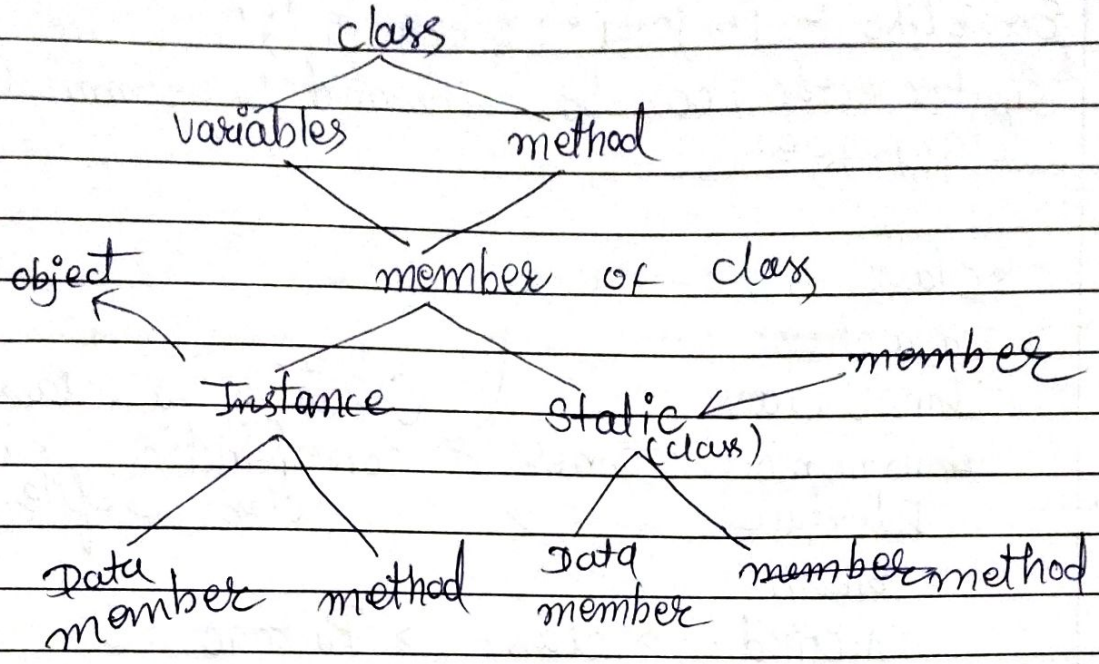
keyword → class Test {
public static void main (String args[])
{
System.out.println("Java is very simple");
}
}



* Out is a static object of printstream class
* Out is a static member of System class

System . out . println ( ) ;
↓ ↓ ↓
class object method.

* without the object, we cannot call the instance method of class.



"Class loading"

- Ideal loading (load all class)
- Lazy loading (load only essential class)

P1.java

```
class A {
}
class B {
}
class Test {
    public static void main(String args[]) {
    }
```

P1.java → A.class, B.class, Test.class

Class loader

* for run java Application (program)

① Javac file_name.java → Compilation

② Java class_name / file_name → Run

lang. :- It is default package in java

PAGE NO.:

Error like :- P1.java:3: error: ';'
Syntax error, can be generated by grammer or language.

- class
- Java class
- Inner class
- Annonymous Inner
- Interface
- Enum

} Creating a .class file corresponding each class file.

method --> class --> Package

* Printing method :-

These are all the instance method of PrintStream class.

- println() :- cursor moves automatic shifted next line
- print() => print the message
- printF() :- formated function
- err.println() :- To print error massage.

- lang :- It is default package in java
- System :- It is built-in java class available in lang. package
- out :- It is an object of printstream, static member of System class
- println
- How to call instance method :- using object

* Escape Sequence character

\n --> new line
\t --> tab space
\r --> carriage return
\b --> backspace
\a --> alarm | beep

System.Out.println("A\n"); o/p ⇒ A\n

System.out.println("\"welcome\""); o/p ⇒ "welcome"

SOP ("ABC\bDEF\rGH");

o/p ⇒ GHDEF

* Access specifiers

public static void main

Access Specifier → <default>, private, protected, public

* If we public the class, we save the file name same as the class name

Pl. Java → Test.java.

Top level class must be public but inner class can be protected or private.

D:
↳ basic
↳ P1.java

PAGE NO.:

class Test
P.S.V.M(S a){

println() :- It is instance method of printstream class

---

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

Why java main class is static ?

* JVM calls the main method by using class. name (for saving the memory)

* Calli Should we call main method using object ?

~~Ans~~ ~~Yes~~ No, it is worst case

In java, main method is public, because JVM can access it within the class or ~~without~~ outside the class.

if we are returen the int value, the main method should int

like ~~int~~

void * :- in java, the main() method does not have return any value.

main method is public because JVM can call the main method from anywhere

PAGE NO.:

public, static ⇒ ~~identifier~~ modifier
we can swap the ~~identifier~~ modifier

*void :- The main method must be public, static and the return type is void.

Parameter
1) Actual parameter
2) formal parameter

formal parameter

```
void add (int x, int y){

}
add(20, 10);
```

Actual parameter

* We can store any type of data in String.

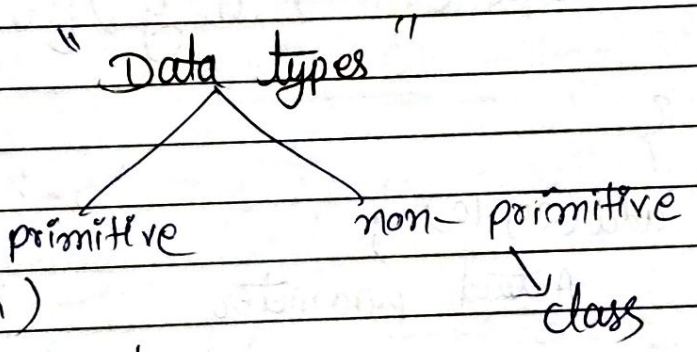
* we can pass any type of value, that is why we declare String

* Valid ~~signature~~ of main method.

(1) Public static void main ~~method~~ (String args[])
(2) public static final void main (String args[])
(3) Public static ~~no~~ synchronized void main (String args[])
(4) Public static strictFP void main (String args[])
{strict floating point}

5. Static public void main (String args[])
6. public static void main (String... args)
// ... var-arg (variable length argument)
7. P S final synchronized strictfp void main (String ... bili)

To process the data.
To store the data, we use variable.



(Built in)
{ byte, float, double, char, int, dar boolean, long, short }

Integer ⇒ byte, short, int, long.
(20, -10, 32764)

Floating point ⇒ float, double
(20.5, -0.5)

char ⇒ Single character
('A', '8') ✓ ('AB') ✗

boolean ⇒ True, false

int *p;
pointer to integer.

Signed (default)
unsigned (explicitly)

PAGE NO.:

Pointer is fixed size.

why java introduce the byte concept.

~~void main(){~~

~~int x;~~ // the type of x is signed integer

~~}~~ each and every

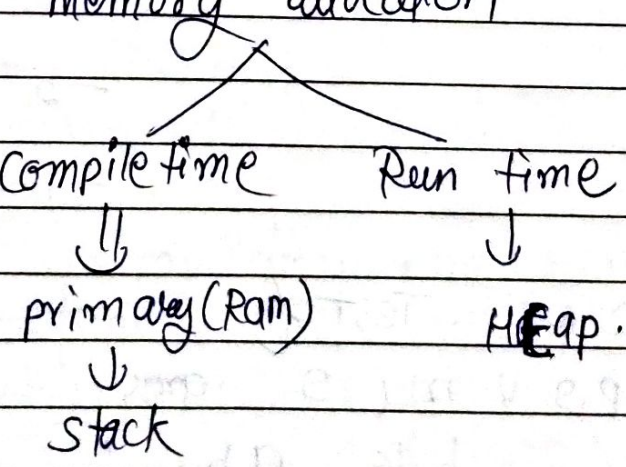
* In java, variable by default signed.

* In java, the garbage value concept is not allowed (not aplicable)

* Architectural neutral in java

"~~Memory Allocation~~"

* Compile time Memory Allocation :-
A compiler is able to resolve, How much memory is required by the variabl at the time of compilation



* Memory allocate at the runtime compilation.

PAGE NO.:

Datatypes

- Byte
- int
- char
- Short
- long

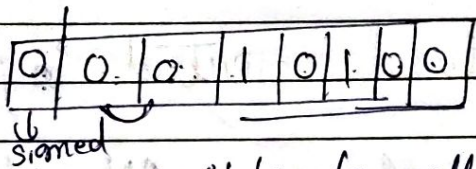
→ char / single letter
→ Integer dat stored.

- float
- double

→ floating point

- boolean → false / true

```
int x = 20;
```



variable is responsible to allocate the memory.

Byte variable

1 byte = 8 bit

-1 (signed)

7 bit

-2^7 to $+2^7-1$

-128 to +127

```
class Test {
PSVm( S... args
    byte a,b,c;
    int a = 20;
        b = 10;
        c = a+b;
    System.out.println(c);
}
```

2000	2001	2002
a	b	c
20	10	30
1-byte	1-byte	1-byte

dt1.Java

```
class TestMain{
  P S v m (S...args[]){
    byte a,b,c; // -128 to 127
    a=20;
    b=10;      } Replace
    c=50;      }
    c=a+b;
    S.o.pln(c);
}
/*
    C.E :- incompatiable type possible lossy conversion
                                from int to byte.
*/
```

error

```
C=a+b;
  ^
```

* cannot asign 32 bit data in 8 bit, it will create error.

c = a+b c = 10+20 c = 30 int byte 8bit ← 32 bit Data loss	(Data loss) Rule :- Type cast

Type casting :-

The proces of convereting one type value to the anothere type is called type casting

syntax :-

```
C=(byte)(a+b);
```

① Implicit
② Explicit

Syntax :- c= data-type (variable).

① Implicit :- Typecasting which is automatically done by the compiler or System.

short 2-byte 16 bit -2^{15} to $2^{15}-1$ (-32768 to 32767)

byte 1 byte 8 bit 2^7 to 2^7 [-128 to 127]

② **Explicit :-**

type casting which is done by programmer forcefully is called explicit

Byte.

-128 to +127 0 to -128

0 to +127 Signed should be in the range

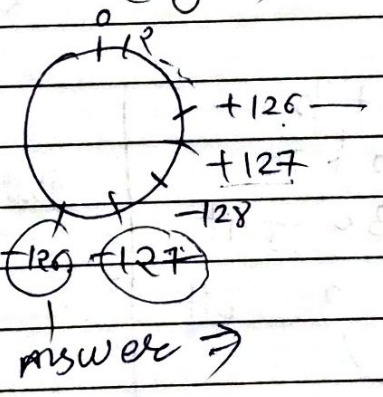
byte a, result

a = 126.

result = (byte)(a+4);

126 + 4 = 130.

result = (byte) 130.



answer ⇒

Rule :-

for use Local variable, first we should initialize variable

In case, int the circle automatic mode If the data is out of range.

PAGE NO.:

Data Types (Defines which type of data, a variable stored)

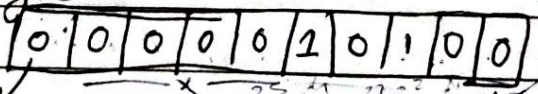
- Primitive
  - Numeric
    - byte
    - Integer
    - Short
    - long
    - (It store Integer data)
    - float
    - Double
    - (floating point data)
  - Non-numeric
    - character (letters) (unsigned)
    - Boolean (true/false)
- Non-primitive
  - class
  - Array
  - Interface
  - String
  - object

	Size :-	sign bit ↓	Range
Byte	1 Byte	⇒ 8 bit - 1, 7bit	-128 to +127
Short	2 Byte	16bit - 1, 15bit	$2^{15}+1$ to -2^{15}
int	4 Byte	32 bit, 31 bit	$2^{31}+1$ to -2^{31}
Long	8 Byte	64 bit, 63 bit	$2^{63}+1$ to -2^{63}.

eg :- int x = 20;

10100

x = 4 byte



Reserved sign bit — Signed Byte

Define variable

byte a, b, c; a=20; b=10; c=a+b;

Create memory block corresponding to each variable

Error :- double cots requried, or not found.

PAGE NO.:

memory allocation of variable is not Continuous.
Access data by using name without double cots.

```
class d{
Byte a = 20
Byte b = 10
    c = a+b;   // The value (Result) of airthmatic op
    S.O.P(c)      is also integer (By default).
```

↳ error [data loss] // it will create error bcoz 32 bit data cannot stored in 8 bit, it will be the problem of dataloss.

Solution:- C = (byte) (a+b);
→ Type Conversion.

Type Casting :-

(1) Implicit ⇒
- when smaller data type Converted into larger data type
- No data loss
- Automatic conversion perform by java compiler

```
int myInt = 10;
double mydouble = myInt ; // Implicit
```

(2) Explict :-
- when larger data converted into smaller data type

n<0 & n=0 & n>0 +a, -b -c.

PAGE NO.:

Operators :-

It is a symbol which perform the operation on operand (data).

Unary :- needs only one operand

Binary :- needs only two operand

Ternary :- needs three operand

Types :-

1. Arithmetic operator (/, -, *, +, %) Remainder
2. Relational operator (>, <, >=, <=, ==, !=)
3. logical operator (&& (AND), || (OR), ! (NOT))
4. Assignment (=)
5. Increment / Decrement (++, --)
6. Bitwise operator (&, |, ^, <<, >>, >>>, ~)
7. shorthand operator (+=, -=, *=, /=)
8. Conditional operator (? :)
9. instanceof

(1) Arithmetic operator :-

S.O.P("7/2:"+(7/2)); 3

S.O.P("7%2:"+(7%2)); 1

S.O.P("-7%2:"+(-7%2)); -1

S.O.P("7%-2:"+(7%-2)); 1

S.O.P("-7%-2:"+(-7%-2)); -1

S.O.P("7.5%2:"+(7.5%2)); 1

* Sign of numerator is the sign of result.

* modulus operator can use floating point value

In floating point whether either is Infinity

PAGE NO.:

S.O.P +(20%7)@//2

S.O.P. +(7/0) // Arithmetic exception

S.O.P (7.5/0) // Infinity

S.O.P (-7.5/0) // -Infinity

S.O.P (7.5%0) // NAN

S.O.P (7%0) // Arithmetic exception

* If floating point divided by 0 it is Infinity
(-floating) point divided by 0 it is -infinity

* floating point modulos by 0 it is NAN.

"(0%7)) 011

int n = 123;
n = n/10;
Removed Last digit in any number

int n=123;
int r = n%10;
if gain, last digit add
get last digit

② Relational operator :-

⇒ Show the relation.
In a result it will give True/false.
If the relation is satisfied // true
If the relation is unsatisfied // false
(20>20) Return Boolean

S.O.P ("20>20 : " +(20>20)) // false

S.O.P ("20>=20: +(20>=20) // True

S.O.P ("20==20: +(20==20)) // true

S.O.P ("20=20: +(20!=20)) // false

we cannot convert boolean into other data type

we ~~cannot~~ compare boolean to boolean only

```
class Test Main{
  P S V M ( S a){
    int a,b,c,d;
    a=10;
    b=10;
    c=10;
    d=10;
    boolean x=a==b==c==d; // incomparable types: boolean and int
  } SOP(x);
```

solution :-

```
boolean x=(a==b)==(c==d);
          True == True
```

x = a==b==c==d

10==10==10==10

x = True == 10

True == 10

* operator precedence :- higher priority first
* operator associative :-
  L → R
  R → L

OP.

precedence "BODMAS" higher priority first, Lower priority second

Associativity L→R, R→L

③ Increment / Decreement :-

++ -- (Auto Typecaste operator)

increase by 1 decrement by 1

int x. x=10

--x; pre decrement

x--; post decrement

++x x++

pre increment	post increment
first increment the value by 1 than assign	first assign the value than increment by 1

" Boolean is a isolated datatype "

PAGE NO.:

(4) Logical operator :—

1. && AND
2. || OR
3. ! NOT

} Join multiple condition

The logical operator is only work on boolean data.

A (a>b)	B (c<d)	A&&B (a>b)&&(c<d)
T	F	F
F	T	F
T	T	T
F	F	F

Return only T/F (A&&B)

Return T only when both condition is True either it returns false

A || B

(a>b) || (c<d)

T
T
T
F

(A || B)

In logical OR condition even when one is true, the Result is true

AND operator OR operator.

A (a>b)	B (c<d)	A&&B (a>b)&&(c<d)	A||B (a>b)||(c<d)
T	F	F	T
F	T	F	T
T	T	T	T
F	F	F	F

```
Class TestMain{
    PSVM (S args[]){
    S.OP(100&&200); error
```

* short circuit behaviour of operators

PAGE NO:

⑤ class Test{
PSVM (S args[])
int a,b,c,d.
a=10
b=20
c=30
d=40
boolean x=(a>b) && (++c<d);

short circuit behaviour

(*
x=(a>b) && (++c<d);
(10>20)
x= false && (++c<d)

false
*/
System.out.println(x) // false
SOP("a:"+a); // 10
SOP("b:"+b); // 20
SOP("c:"+c); // 31
SOP("d:"+d); // 40.

* If we want to check the second condition also, we simply use '&'.

* when we use "&&" operator, then only first condition is checked, than return answer.

* Syntax:-

Both operators "AND" and "OR"

class Test {
PSVM (S args[]){

```
boolean x = (a>b) && (++c<d) || (++a<c);
/*
    x = (a>b) && (++c<d) || (++a<c);
        (10>20)
        false && (++c<d) || (++a<c);
                 false || (11<30)
                 false || true
                 True.
*/
```

* Short circuit behavior of logical operator :—
the ability of certain logical operators (like AND and OR) to potentially stop evaluating the expression as soon as the overall result is determined.

- Logical AND (&&) :—
In an expression like A && B, if A is false the entire expression is false, and B is not evaluated.

- Logical OR (||) :—
In an expression like A || B, if A is true, the entire expression is true, and B is not evaluated.

(Ternery operator)

Conditional operator :-

provides a concise way to express a conditional (if-else) statement. It is the only operator in java that takes three operands

* we cannot print "printing statement" in conditional operator
* line execution " used conditional operator;

Syntax :-

```
(boolean expression/exp1) ? exp-2 : exp-3;
```

```
class TestMain{
    psvm(S---args[]){
    int n=10;
    String s=(n>10)? "Indore" : "Pune";
    System.out.println(s);
    }
}
```

If value is True ↓ If value is false ↓

```
class Testmain{
psvm(S args[]){
    int n=10;
    byte s=(byte)(n>10)? 200 : 100;
    System.out.println(s);
}
```

Package "

import java.util.Scanner

Instance methods of class

- → nextByte() Read Byte value
- → nextFloat() Read Float value
- → nextShort() Read short value
- → nextLong() Read Long value
- → nextDouble() Read double value
- → nextBoolean() Read Boolean value
- → next() Read single word string
- → nextLine() Read whole line

firstly we create the object of Scanner class, than call instance method.

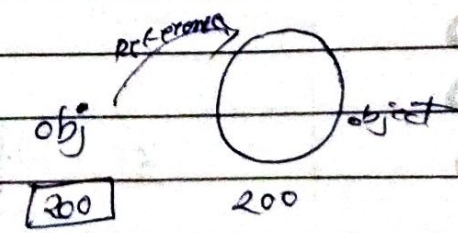
* Creating object ⇒

① new A();

new → keyword, A() → classname

② A obj = new A();

A → reference object, new A() → object



Memory occupied by object

data read ⇒ input stream
supply data ⇒ output stream

stream ⇒ series of bit

PAGE NO.:

* The object without reference (Annonymous object)

⇒ new A();

① Reference Approach	Annonymous Approach
for using the object multiple times	for using the object only single time at the time of creation

```
Scanner sc;
sc = new Scanner();
{ 1000
  =
  =
  =
}
} → end
```

Heap.

sc → 4000

Destroy after whole execution

```
new Scanner();
  =
  =
  =
} 1000;

→ end
```

Heap.

⇒ destroy at any time

* One object can have multiple references.

Program :-

```
import java.util.Scanner;

class Addition{
public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter first integer value");
    int a = sc.nextInt();
    System.out.println("enter second integer value");
    int b = sc.nextInt();
    int result = a+b;
    System.out.println("Addition : "+result);
}
}
```

Bitwise operator :-

applicable only integer data

[&, |, ^, <<, >>, ~]

n & n = n
n | n = n
n ^ n = 0

```
int x = 15 & 12;
   1 0 8 4 2 1
&  1 1 1 1      8+4 = 12
   1 1 0 0
   ---------
   1 1 0 0
```

(conditional statement)

Control Statement :-

if else, if, nested-if-else
Switch statement

loop
--- do while
--- while
--- for

IF ⇒

Syntax :-

① if (condition/boolean---expression)
    statement --- 1;

② if (condition/BE){
    statement --- 1;
    statement --- 2;
}

③ if (condition/BE)
    statement --- 1;
else
    statement --- 2;

④ if (condition/BE){
}
else
{
}

Rules :-

if के just bad else statement आना चाहिए।

Bich में कोई statement नहीं आना चाहिए।

-21 -21%2 = -1

PAGE NO.:

write a program to enter an integer number and check it is even or odd

```
import java.util.Scanner;

Class Testmain{
    PSVM( S Aogs[]){

    Scanner sc = new Scanner(System.in);
    S.O.P.ln ( " enter number n ");
    int n = sc.nextInt();
    if (n%2==0)
     S.opt("EVEN");
    else
     S.O.pln("ODD");
    }
}
```

"String object"

"HELLO"
01234

charAT (0) 'H'
{ method of string class }

PAGE NO.:

Switch Syntax :-

```
{ Switch ( expression / value ){
    Case label-1 : Statement -1;
                   Statement -2;
                   break;
    Case label-2 : Statement --2;
                   break;
    default : default-statement;
}
```

expression and label - must be same type.

* Case के अंदर If, else, switch case use ~~कर सकते~~;
* Case label should be unique.
* Case label should be Constant.
* Case can be character constant 'A'.
* we can pass string constant as well " Intore "; Supports only in Jdk 7
* charAT(0) is used to read the character value
* " '1', '*', '-', '+' " all are character values.

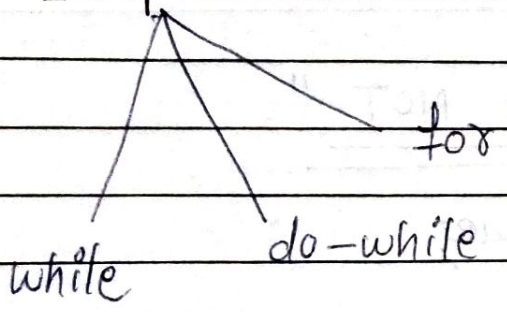
DRY KISS { Two formula's to code } and Run

In one block of code, we cannot intialized same name variable multiple times.

Switch (x) :- In this switch block, the default and max value is int,

valid switch expression value:- byte, short, int, char, String.

* "Loop":-



* To execute the statement and block of code, multiple times.

To solve the iterative problem, we use Loop.

loop → initialization
→ condition
increment / Decreement
(we can use '*' and '/' also for increment & decreement)

① while Syntax:-

→ initialization;
while (condition / BE)
{
——;
——;
++/--;
}

```
class Test{
SOP(----
int i=1;
while (i<=5)
{
SOP("Java");
i++;
}.
```

"Logics"

To print Java 5 times

```
i = 1;
while (i != 6) {
    SOP("Java");
    i++;
}
```

```
i = 5;
while (i >= 1) {
    SOP("Java");
    i++;
}
```

"Prime or NOT"

```
Boolean status = True;
int n; i = 2;
while (i <= n/2) {
    if (n%i == 0) {
        Status = false;
        break;
    }
    i++;
}
if (Status true && n > 1)
    S.O.P("Prime");
else
    S.O.P("Not prime");
```

"Logic fabonnaci Series"

PAGE NO.:

```
int a,b,c;
a = 1 ;
b = -1 ;

c = a+b;
s.o.p(a);
s.o.p(b);
i = 1;
while(i<=(n-2)){
c = a+b;
s.o.p ln( c);
a = b;
b = c;
i++;
}
```

```
a = -1;
b = 0; 1
while(n!=0){
c = a+b;
s.o.p(c);
a = b
b = c;
n--;
}
```

"LOGICs"

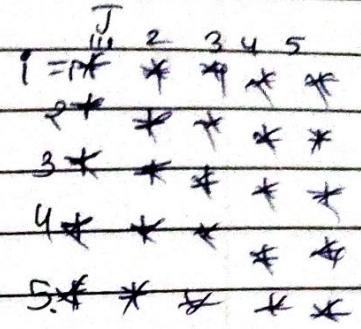
A number is Palindrome or not

```
int rem;          // remainder
int rev = 0;      // reverese digit
int original = n;
while (n>0){
    rem = n % 10 ; // last rem
rev = (rev *10) + rem;
  n = n/10 ; // to remove last digit
}
 s.o.p("rev :" +rev);
if( rev == original){
   so.p (" Palindrome No.");
}
else{ s.o.p(" not a Palindrome")
}
```

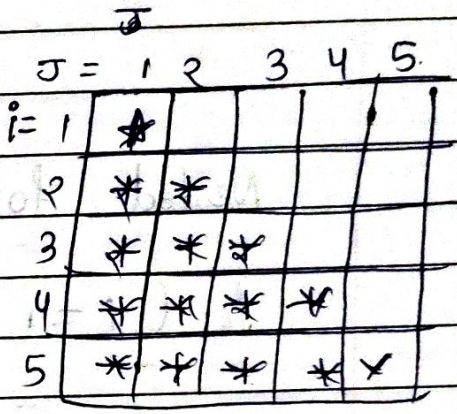
number is Armstrong or not

```
int Sum = 0;
int rem;
int original = n;
while( n!=0){
rem = n%10 ; // remainder
Sum += rem * rem * rem;
  n = n/10; // remove last digit
}.
if (sum == original){
  s.o.p ("number arms trong")
}
else {s.o.p (" not a armstrong")
};
```

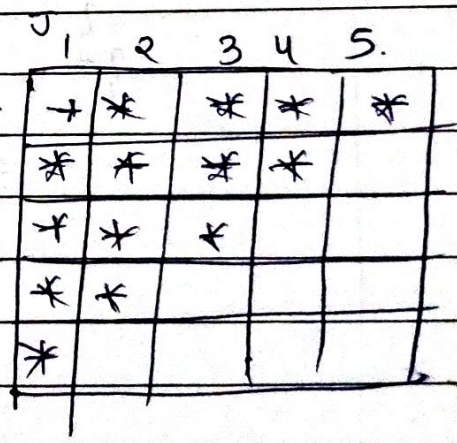
①
```
for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++){
        S.o.print("*");
    }
    S.o.println();
}
```



②
```
for (int i=1; i<=5; i++)
{
    for (int j=1; j<=i; j++)
    {
        S.o.p("*");
    }
    S.o.println();
}
```



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



(i-1)

5-1 ④

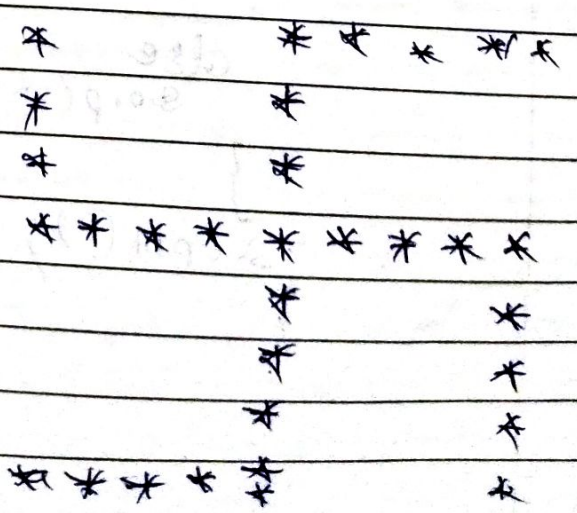
PAGE NO.:

	1	2	3	4	5	J.	i
					*		5
				*	*		4
			*	*	*		3
		*	*	*	*		2
	*	*	*	*	*		1

④
```
for (i=5; i>=1; i--){
    for( space=1; space<i; space++)
    {
        s.o.p(" ");
    }
    for (j=5; j>=1; j--){
        s.o.println("*");
        s.op.println();
    }
```

	J=1	2	3	4	5	6	7	8	9
i=1					*				
2					*				
3					*				
4					*				
5	*	*	*	*	*	*	*	*	*
6					*				
7					*				
8					*				
9					*				

⑤
```
for (i=1; i<=9; i++)
{ for(j=1; j<=9; j++)
  { if ( i==5|| j==5)
      s.o.p( "*");
    else
      s.o.p(" ")
  }
  s.oprintln();
}
```



PAGE NO.:

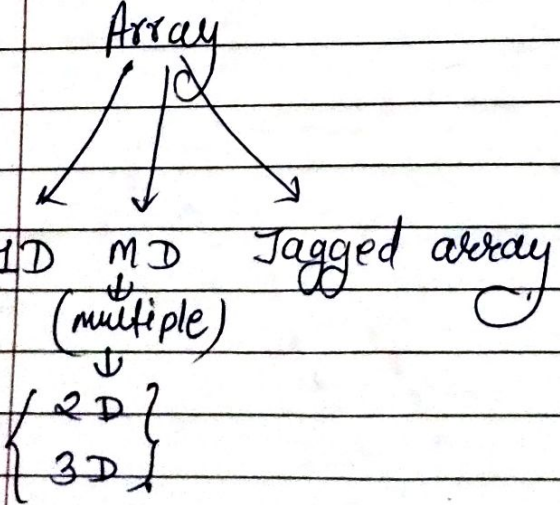
"Array"

Array — why
Array — what
Array — How.

why ⇒ • used to store large amount of data in single unit
• Data must be same type.

what ⇒ • A collection of Homogeneous type of data.
* when the quantity of data is already advance./ known than we use array.
* Array is fixed in size.

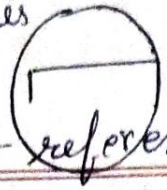
How :-

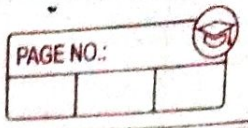


① 1 Dimensional array :-

A variable of class Type. is a reference variable

class → Scanner sc; ← reference.





Size must be integer

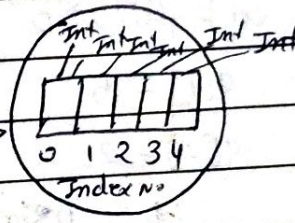
Syntax :—

datatype array_name[] = new data_type[size];

(int arr[] = new int[10];)

or

int arr[];
arr = new int[10];

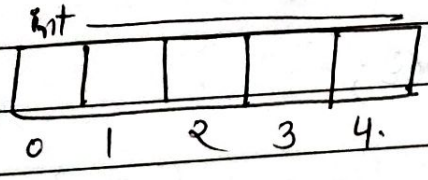


arr [2000]

2000

Indexing :— "starts from 0 to 1".

int arr[] = new int[5];



int.

float arr2[] = new float[5];

reference.

arr.getclass();

I + class.

Scanner sc = new Scanner();

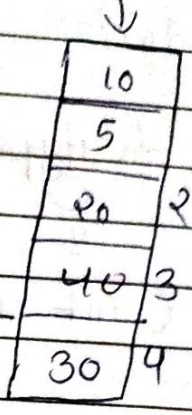
sc.getclass();

java.util.Scanner;

- Accessing data elements :-

* using index number →

```
int arr [] = new int [5];
arr [0] = 10;
arr [2] = 20;
```




```
for ( int i = 0; i < 5; i++){
    System. out. println( a[i]);
}
```

a[i] = a[0] ⇒ 0
a[1] = 5
a[2] = 2.

"i ⇒ index number"

* using loop ↑

Program :-

```
import java.util. Scanner;
Class Main{
    P S V M ( S A [])
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt ();
    int arr [] = new int [n];
    for ( int i=0; i<n; i++) {
        System.out.println ("Enter "+(i+1)+" element Number");
        System.out.println (" a[i]");
        arr [i] = sc.nextInt();
    }
    System.out.println (" Give Data... ");
    for ( int i=0; i<n; i++)
        System.out.println( arr[i]);
}
```

PAGE NO.:

Sorting :-

```
int arr[] = {10, 7, 15, 8, 3, 4, 3, 6};

for(int i=0; i<arr.length-1; i++)
{
  for(int j = i+1; j<arr.length; j++){
(6).    int temp;
    if (arr[i] > arr[j]){
      temp = arr[i];
      arr[i] = arr[j]
      arr[j] = temp;
    }
  }
```

---

* Binary Search *

Binary search only applicable on sorted data.

a → | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
0 1 2 3 4 5 6 7 8 9

Low index → 0
mid → 4
High index → 9

midindex = (low + High)/2;

⇒ (0+9)/2

mid. ⇒ 4.

ele ⇒ 90;

Binary Search algorithm

```
while (low ≤ high)
{mid = (low + High) / 2 ;
 (IF (a[mid] == ele)
   break;
 else if (ele > a[mid])
   low = mid+1 ;
 else
   High = mid-1 ;
 }
 if (low > high)
   S.O.P ("ENF");
 else
   S.O.P ("EF");
```

---

* Insertion in array.

```
int ar[ ]=
 S.O.P ("enter size
int n=SC.nextInt();
int arr[] = new int[ n+1];
S.O.Pln ("enter"+n+"element");
for( int i=0; i<n; i++)
  ar[i] = SC.nextInt();
S.O.P("enter position");
int position = SC.nextInt();
S.O.P ("enter element");
Int element = SC.nextInt();
```

arr.length ⇒ it return the total no. of rows when we apply on 2D array

PAGE NO.:

#:- 2D Array :-

2 dimension array :-
collection of one D array.

Syntax :-

row ↓ column ↓

```
int arr[][] = new int[3][3]
```

r \ c	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

3×3.

00	01	02
10	11	12
20	21	22.

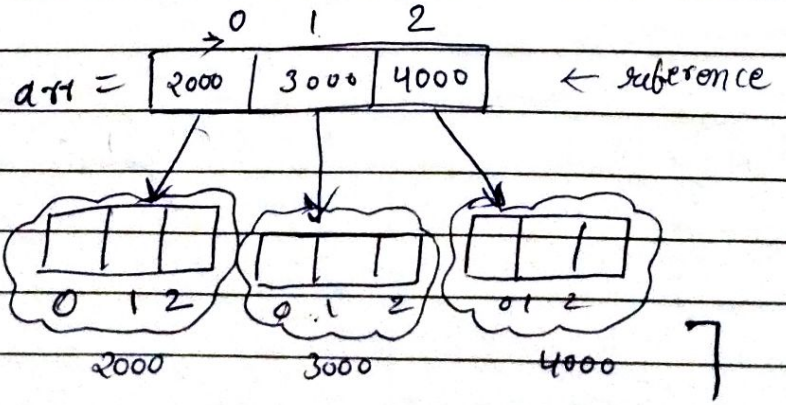
Indexing.

```
for (int r=0; r<3; r++)
{
    for (int c=0; c<3; c++)
    {
        arr[r][c] = sc.nextInt();
    }
}
```

12
arr[1][2]

"Memory mapping"

```
int arr[][] = new int[3][3]
```



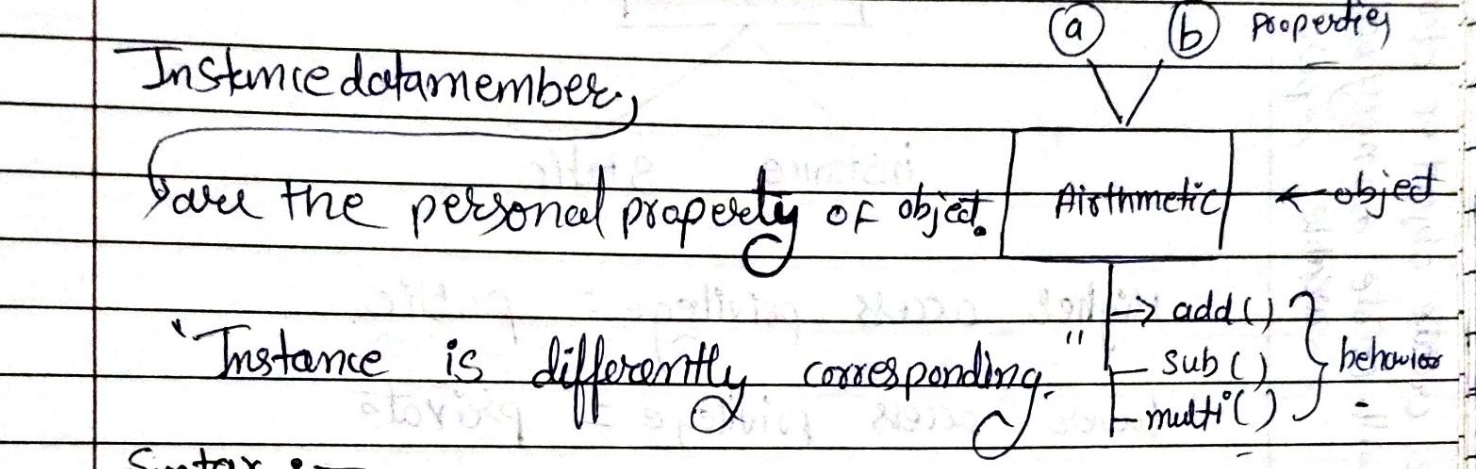
arr[0].length

* JAVA *

PAGE NO.:

OOPs :-

* If i have to represent a object, i will have to create a class.



Syntax :-

Class-Name obj(Ref.) = new class-Name ();

Ex:- Arithmetic obj = new Airthmetic ();

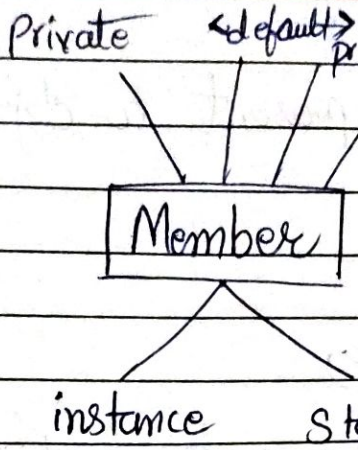
```
class Airthmetic{
    int a;    // instance datamember
    int b;

    void add(){          // instance method
    ===
    }
```

"There is no global variable in java."

Class Type variable is reference variable

PAGE NO:

Arithmetic obj = new A



// Access specifiers.

Higher access privilege = public.

Lower access privilege = private

1. Private
2. <default>
3. protected
4. public

Higher security.

weaker security

Public

```
Class Arithmetic {
Public int a;
public int b;

public void add() {
    S.o.p(a+b);
}

Class MainTest() {
p s v m(s---args[]) {
Arithmetic obj = new Arithmetic(); // object of class....
obj.add();
```

when we create object of class it will create memory of all instance variable separately

Instance means behaviors of ~~class~~ object...

Instance data member = stored data.

PAGE NO.:

* Datamember :- variable of class

* a member can access a member directly.

* Object cannot be access directly the data

~~Arithmetic = obj~~ * ~~Publicly access~~ *

Arithmetic obj = new Arithmetic ();

obj.a = 20;

obj.b = 10;

obj.add ();

}

current object

Arithmetic obj2 = new Arithmetic();

obj2.a = 40;

obj2.b = 30;

obj2.add();

~~Public~~ private

class Arithmetic {

private int a;

private int b;

Public void add(){

s.o.p (a+b);

}.

Public void setdata(int x, int y)

{

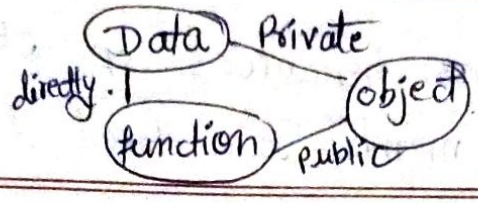
a=x;

b=y;

}

class main{

P S V M (S ...args[])



PAGE NO.:

```
Arithmetic obj = new Arithmetic();
obj.SetData(20,10);
obj.add();
}
```

```
Class Addition{
private int a;
private int b;
public void add(){
S.O.P("Result :" +(a+b));
}
public void setdata(int x, int y){
a=x;
b=y;
}
}
class Testmain{
P S V M (S args[]){
Addition obj = new addition(); // object
obj.setdata(10,20);
obj.Dot add();
}
```

Keeping data private is recommandable approach---
current obj, जिससे method call हो रहा है
primary priority gives two local variable.

This" is used to reference current object of class.

Class Addition{

private int a; // Instance member
private int b;

Public void add(){
s.o.p(a+b); // (this.a + this.b) Also.....
}

public void SetData(int a, int b){
this.a = a; // local variable
this.b = b;
}

class MainTest{

PSV M(S...args){

Addition obj = new Addition();
obj.SetData(20,10);

* Setter is used to update the object's property. (int a; int b;)

PAGE NO:

```
class Addition{
    private int a;
    private int b;

    public void setData (int a, int b){
        this.a = a;
        this.b = b;
    }

    Public void add(){
        S.o.p(a+b);
    }
}

class Main Test{
    P S V M (S...arg[]){
        Addition obj = new Addition();
        obj.setData(20,10);
        // If we want to update existing object data;
        obj.setB(5);
        S.o.p(obj.getB());
    }
}
```

```
Public void setB(int b)
{
    this.b = b;
}

Public int getB(){
    return b;
}
```

* getter is used to reetiven the value. (get value)

PAGE NO.:

Method :—

1) Instance Method :— Belongs to object [Behaviour of object]

2) Static Method :— Belongs to Class [Behavior of class]

Call by value

Call by reference.

"Method Overloading"

PAGE NO.:

Overloading = why, what, How.

Polymorphism :- one thing has different implementation

- compile Time [overloading]
- Runtime [overriding]

* Method overloading :- defining multiple method with the same name whose parameter are different.

NOTE :- In case of method overloading the return type doesn't matter.

* "Why it is compile Time polymorphism" :-

JAVA compiler resolve that when you call the function, for what type of data value and How many parameter the function Have.

In case of method overloading :-
first compiler go for perfect matching, if it is found, it go for suitable matching, if suitable is not found, then it generate error.

① Perfect matching, (2) Suitable matching (3) generate Error

* If there are two suitable method it generate ambiguity error; ← compiler.

PAGE NO.:

```
Class Overloading {
public void m1(){
    S.o.p("m1() - default");
}
public void m1(int x){ ← formal parameter.
S.o.p("Interger-version");
}
public void m1(float x){
    S.o.p("float version");
}
}
```

```
class main{
PSVM(S...args[])
overloading obj = new overloading;
obj.m1();
obj.m1(20);
obj.m1(20.5F);
```

[one dimension array]

* Var-arg :- Variable length argument

var-arg [zero or Any length of arg.]

```
Class Test{
    Pv m1(int... x){
    int sum=0;
    for(int element : x)
    sum = sum + element;
    S.o.p(sum);
}
}.
Class Main{
Test obj = new Test();
obj.m1(20,10,5,30);
}
```

when we pass multiple arg. in single method defination, we can use var-arg

when, we pass DIFF. no. OF As parameters compiler creates a new array every time corresponding to each changes.

when we use vararg in and non varArg arg.
in the same method, The varArg must be the last one Argument

```
Class TestVarArg{
Public int add (int... X){
    System.out.println(X.getClass().getName());
    int sum=0;
for (int i=0; i<x.length; i++){
        sum= sum+x[i];
    }
    return sum;
}
Class Main{
    P.S V M (String args[]){
TestVarArg
Class obj = new Class();
int result=obj.add(20,10,5);
S.O.P("Addition:"+result);
    }
}
```

The Constructor is responsible to Initialize the object.

PAGE NO.:

(It construct the memory for object)

Constructor :-

Why what How

Constructor is a special member of class because of following reasons

(1) The Name of constructor is same as the name of class

(2) There is no explicit return type of the constructor

(3) Constructor cannot be static

(4) Constructor call automatically when the object is created.

why we use constructor.

NOTE:- wh constructor is used initialize the object at the time of object creation.

Types.

(1) Default Constructor

(2) Parameterized Constructor

Types of constructor depends on How many ways to create obj.

No. of Constructor = Types of ways to create object

PAGE NO.:

"Constructor Overloading"

```
Class A{
Public A(int x){
  S.o.p("i-version");
}
Public A(boolean x){
}
Public A(String s){
}
public A(){
}
```

```
new A();
new A(100);
new A(true);
new A("Hello");
new A();
```

* There is NO Copy Constructor, java uses, the Cloning method.

* with the help of this() method, we can call a constructor using another Constructor.

* this() // this-call.

obj.getclass() → part of reflective API

PAGE NO.:

```
Class Test{
    public Test(int x, int y){
        this(x); // this. call   - (It must be first statement)
        S.o.p ("Int Int version");
    }

    Public Test(int x){
        this();
        S.o.p("Int version");
    }

    Public Test(){
        S.o.p("Default version");
    }

Class Main{
    P S. v. M(S... args []){
        Test obj = new Test(10,20);
    }
```

constructor

object ← constructor.

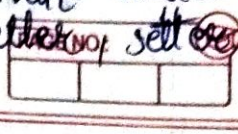
new Test(10,20);

object

* when you call this(), it must be your first statement in your constructor.

POJO :- A class has all properties private variables and Constructor, getter, setter.

"Plane old JAVA Object"



Recursion :—

When function call itself, it is called recursion.

- compiler generate error.

error :— Recursive Constructor invocation.

```
Class A {
  public A(int x){
      this();
  }
  public A(){
      this(200);
  }
}
```

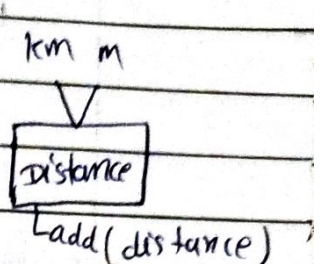
← error

Stack OverFlow, when JVM stops it.

Concepts :-

(1) Passing object

(2) Returning object



"Passing Object"

```
Class Distance{
    private int km;
    private int meter;
    public Distance(int km, int m){
        this.km = km;
        this.meter = meter;
    }
    public distance(){
        km = meter = 0;
    }
    public int getkm(){
        return km;
    }
    public int getmeter(){
        return meter;
    }
public Distance add(Distance d){
    Distance temp = new Distance();
    temp.km = this.km + d.km;
    temp.meter = this.meter + d.meter
```

Time
Hour Minute Second:

PAGE NO.:

```
if (temp.meter >=1000)
{
    temp.kmH++;
    temp.meter = temp.meter %1000;
}
  return temp;
}

Class TestMain{

  PSVM (S..args[]){

  Distance d1 = new Distance(5,700);
  Distance d2 = new Distance(2,600);

  Distance result = d1.add(d2);

S.O.P(result.getkm()+":"+result.getmeter());
}
}
```

variable ⇒ local.

Datamember : variable at the class scope

PAGE NO.:

Static member :- Belongs to class.

(a). Data member

(b). method.

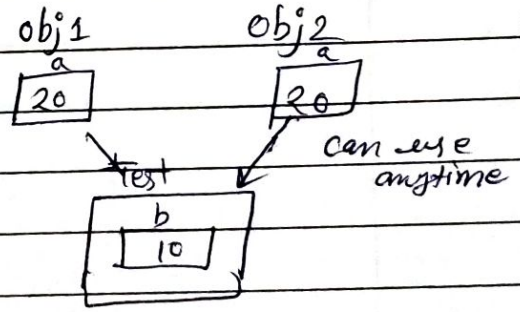
Instance & Static Data member :-

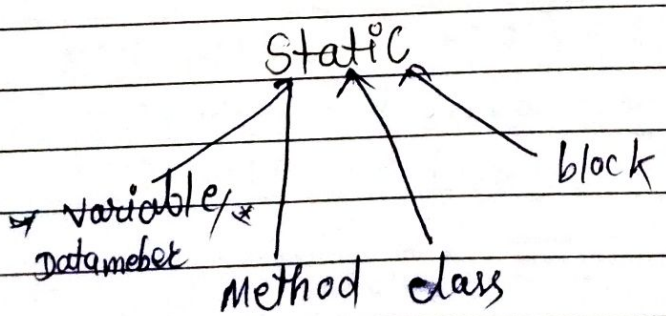
Data :-

Instance data :- Separate corresponding to each object

static data member :- common (sharing for all object)

```
class Test{
private int a=20;    // Instance
private static b=10; // Static (common).
}
```





In java, the static keywords, modifier, we can make Datamember, method, class block static.

method

* Instance method can use instance member or (call) static member.

PAGE NO.:

* Static variable create a single copy corresponding to the class.

* we cannot use (call) instance member in static method.

* Non-static (a) member cannot be referenced in static-context.

```
class Test

private int a=20;
private static int b = 10;

public void increment(){
        ++a;
        ++b;
}

public static void display(){
  S.o.p(a);   <-- error :- Non static(a) member cannot be referenced in static-context
  S.o.p(b);
}
```

Static-context
- static method
- static block

Static || errors :-

error :-

Non-static method cannot be referenced from a static context.

* we want to access a non-static block from a static block object. so, the Compiler is given the error.

static → Static

Instance → Static Instance.

PAGE NO.:

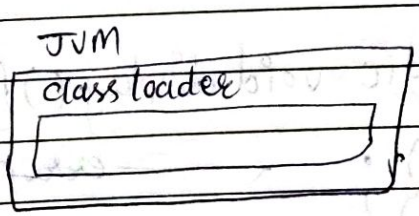
When the static member allocate memory?

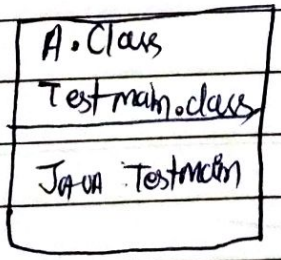
At the time of class loading, static variable Create a memory.

* How to create static block?

```
Class A{
    Static{        // static block.
        ≡
    }
}
```

* The classloader is responsible to loads all .class file in memory when they are in use.





* Static block, first calls, when the class loads.

from the Jdk 1.7 version, we cannot run the application without the main class.

before the Jdk. 1.7 and onwards, we can run the application without main method.

PAGE NO.:

```
Class A{
    static{
        S.o.p("Block-1");
    }
    P S V M(String args[]){
        S.o.p("main executed");
        new A();
    }
```

output:

Block-1
main executed

when, there are numbers of static block in class.
The execution of static block is in same order in which they are written.

* Before constructor, if you want to execute something, than you can use initializer block.

Class :-

* we cannot make outer class static class.
* only inner class can be static class.

SpringBoot
Hybernet (Backend)

PAGE NO.:

what is the differen in char Array or String :-

String in java :- — why
— what
— How

~~String is immutable~~

java.lang.String
[lang. package]

String :- In java, String is a build-in java class which is immutable.

* immutable (that cannot be changed)

* immutability :- If we try to make changes in existing string object, than changes is not going to take place in existing object, java will always create new string object.

[corresponding to each and every changes, you will get a new object (java will create new object.)

How to deal with string ?

* Java.lang.String
  - → string ()
  - → String (string)
  - → string (char [])
  - → String [byte [])

PAGE NO.:

* " " when we write anything in " " than java refers it string object.

* Initialization of string class object.

① String s1 = new String ("Hello");

② String s2 = "Hello";

String object. By

String literal By.

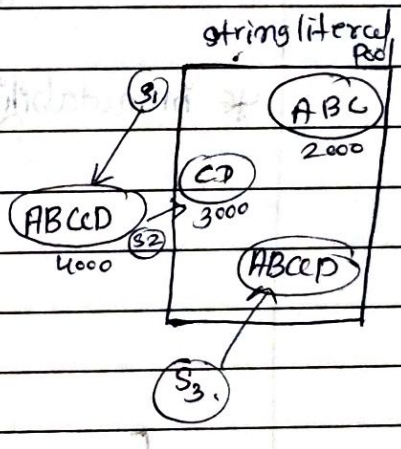
* String s2 = "Hello";

s2.length() ⇒ 5.

object. { "Hello".length() ⇒ 5.

* String literal pool

check object availability.



String s1 = "ABC";

String s2 = "CD";

s1 = s1+s2;

String s4 = "AB";

s1 = s1 +s2;

= "ABCCD";

s1 = "ABCCD";

s3 = "ABC" + "CD";

s3 = ABCCD;

int a,b; int c;

a = 20;

b = 10;

c = a+b

↑ Runtime

Store ⇒ Heap

c = 20+10

↓ compile Time.

Store in literal pool.

* when we perform runtime operation on string it goes to Heap memory

* GC concept is not aplicable for literal pool it only aplicable for Heap memory.

Assignment :- String methods.

PAGE NO.:

In String indexing starts from 0

String Methods. ⇒

(1) int length() method :- This provides the total count of characters in the string.

Ex :- String str = "Hello, World!";
S.o.p(s. length());

output ⇒ 13.

(2) charAt(int i) method :- This method returns the character at ith index.

Ex :- String str = "Hello, World!";
S.o.p(S.charAt(7));

output :- W

String

(3) Substring(int i) method :- This method returns the Substring from the ith index character to end.

Ex :- String S = "Hello, World!";
S.o.p(S. Substring (7));

Output :-
World!

(4) String Substring (int i, int j) method :- This method returns the substring from i to j-1 index.

Ex :- String S = "Hello, WORld!";
S.o.p (S. substring(7, 12));

output :- WORld

(5) String Concat (string str) method :—
This method appends the given string to the end of the current string.

Ex :— String S = "Hello, World!";
S.O.P (S. Concat ("!!!"));

(6) int indexOF (String S) method :—
This method returns the index within the string of the string of the first Occurrence of the specified string. If the specied String S is not found in the input string, the method returns (-1) by default.

Ex :— String S = "Hello, World!";
S.O.P (S. indexOF ("World"));
output :— 7

(7) int indexOF (String s, int i) method :—
This method returns the index within string of the first Occurrence of the specified string, starting at the specied index

Ex :— String S = "Hello, World!";
S.O.P (S. indexOF ("l", 4));

output :— 10.

(8) int LastIndexOF (String S) Method :—
This method returns the index within the string of the last Occurrence of the specified string. If the specified S is not found in the

input string, the method returns (-1) by default.

Ex :- String s = "Hello, World!";
s.o.p (s.LastIndexOf("L");

output :- 10

9] boolean equals(Object otherobj) method :-
This method compares this string to the specified object.

Ex :- String s = "Hello, World!";
s.o.p(s.equals("Hello, World!"));

Output :- True.

10] boolean equalsIgnoreCase(string anotherstring) method :-
This methods checks if two strings are equal, without considering letter case.

String s = "Hello, World!";
s.o.p(s.equalsIgnoreCase("hello, World!");

output: true,

11] int CompareTo(string anotherString) method.

This method compare two string lexicographically (according to unicode)

Ex :- String s = "Hello, World!";
s.o.p(s.CompareTo("Hello, JAVA!");

W - J
87 74
-13

output => 13

12] int compareToIgnoreCase(String anotherstring) method:-
This method compares two string lexicographically, ignore case considerations.

Ex:- String S = "Hello, World!";
S.o.p(S.CompareToIgnoreCase("hello, java!");

output :— 13.

13] String toLowerCase() method:-
This method converts all the characters in the string to lowercase

Ex:— String S = "Hello, World!";
S.o.p(S.toLowerCase());

output:-hello, world!;

14] String toUpperCase() method:—
This method converts all the characters in the string to UpperCase..

Ex:— String S = "Hello, World!";
S.o.p(S.toUpperCase());

output HELLO, WORld!.

15] String trim() method:—
This method returns the copy of the string by removing whitespaces at both ends. It does not modify the whitespaces characters present between the text.

Ex :- String s = "Hello, Trim! "; " Hello, Trim! ";
S.o.p(" ' " + s.trim() + " ' ");

output :- 'Hello. trim!' .

10] String replace (char oldchar, char newchar) method-
This method returns a new string where all instances of oldchar are replaced by newchar.

Ex :-
String s = "Hello, world!";
S.O.P (s. replace('l', 'x'));

output :- Hexxo worxd!

17] boolean Contains (char Sequence sequence) method.
This method returns true if string Contains the given string.

Ex :- string s = "Hello, World!";
S.O.P (s. contains ("world!"));

output :- True.

18] char[] tocharArray[] Method :-
This method converts the string into new characters array.

String s = "Hello";
char chars[] = s.tocharArray(); → output
for (char c : chars){ H e l l o
S.o.p(c + " ");

* one object has multiple references *

* Object class is the parent class of all class in JAVA

PAGE NO.:

Concept :-

```
String s1 = "A";
String s2 = new String("B");
String s3 = "A";
```

SCP	Heap
s1 → A	s2 → B
2000	3000
s3 → B	

int a=20, b=20;

a==b Check content

String reference :- s1==s3 true

s1==s2 false Check address.

Method :-

(1) equals() "Compare string content"

s1.equals(s2) → boolean (result)

This equals method derives from object class.

```
class A{
}
A obj = new A();
S.O.P(obj);
    obj.toString(); → belongs to object class.
```

when we print any object directly JVM calls to.string method.

To.string method can call by any object in java

(2) equalsIgnoreCase() :-

It is not case sensitive, it compares the different cases string.

* HashCode is basically used for fast accessing / retrieving the object.

PAGE NO.:

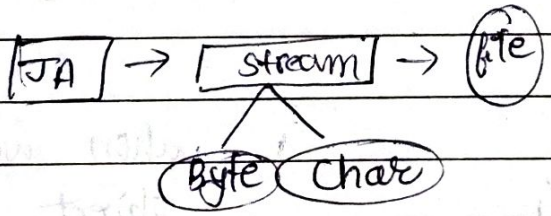
Concepts :-

String $s_1 = "A"$;
String $s_2 = "B"$;
String $s_3 = s_1 + s_2$; "A"+"B" → "AB"
String $s_4 = "AB"$;
String $s_5 = s_1 + "B"$; → "A"+"B" → "AB"
String $s_6 = "A" + "B"$; "AB"

SCP	Heap
s_1 → A	AB
s_2 → B	s_3
AB ← s_4	AB ← s_5
s_6	

← Heap create new object [

* Stream ⇒ Sequence of bytes
Byte stream.
If you want to transfer the data from java application to file, use stream.
It is like a pipeline.



String (byte [ ]) → Constructor.

(1) Byte b[ ] = {65, 66, 67}.

String s_3 = new String(b);
S.o.p(s_3);

Output :- ABC.

{ Byte array :- converts the data to string }.

s4.length(); method
ch.length : property

Constructor.

~~Constructor~~ String (char[])

```
char ch[] = {'A', 'B', 'C'};
String s4 = new String(ch);   { convert char to string }
S.O.P(s4) = "ABC";
```

Methods :—

```
String s1 = "Hello";
```

→ toUpperCase(); // HELLO : string (store in Heap)
ex:- s1.toUpperCase();

→ toLowerCase(); //hello : string (store in Heap)
s1.toLowerCase();

→ length(); : int

→ trim(); : String

→ Contains(string); : boolean.

→ CompareTo(string) : int

0 (same) | +ve (current big) | -ve (formal parameter big.)

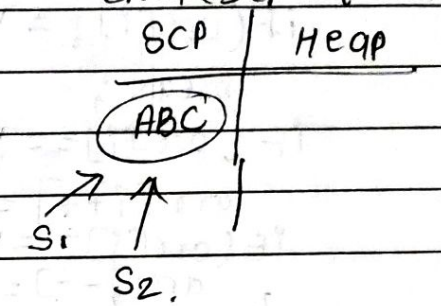
In java, every immutable class is thread-safe.

one thread-safe at a time, mechanism.

intern method :-

```
String S1 = "ABC";
String S2 = new String("ABC").intern();
```

check SCP before Heap memory




```
S.O.P.(S1 == S2) //true.
```

StringBuffer :-

[Buffer :- temporary memory]

- StringBuffer is a built-in class in java. which is mutable.

[changes in same object].

- To create the object, only use new keyword.
- StringBuffer class is threa-safe, [Synchronized].
- Mutable but thread-safe.

Syntax :-

```
StringBuffer sb = new StringBuffer("Hello");
sb.append("world"); //sb :-("Helloworld");
```

(changes in same object)

:- sb.

StringBuffer :- By default 16 character.

when to use string & stringbuffer :-

whenever you want frequently changes in string data, go for stringBuffer.

String

S1.equals(S2) //

// true

StringBuffer.

S1.equals(S2)

// false

Class object {

boolean equals(object){

= location check

}

class String {

PS. equals ( ).

// content check

}

override the parent class

Converts location to content

that why it gives true

class StringBuffer {

Sb equals(){

// == check location

}

Donot override the parent class.

it give false

- use override, when we want to change the implementation of parent class.

- equals() methods is not override that's why it gives false. (string buffer)

- String class check content in equals()
- StringBuffer check location location in equals().

StringBuilder :-

- Mutuable + Don't want thread safety.
- performance releatively fast.
- .equals() method is not override in StringBuilder.

- Mutable but Not thread-Safe.

"Inheritance"...

when object of one class acquire the properties of another class.

"By using existing thing, to develop something new. but these should not be any changes in existing thing"

Extends keyword is used to inherit the class features.
'extends"

Code reusability → Inheritance

Super class → Class J5{
Parent class
V(){
}
X(){
}
}

SubClass/ → Class J7 extends J5
Childclass. {
camera(){
}
}

Class Main{
P.S.V.M(S..args
J5 i5 = new J5();
i5.V();
i5.X();
J7 i7 = new J7();
i7.V();
i7.X();
i7.camera();

Security must be tight in case of Inheritance.

PAGE NO.:

* when we create static block in both classes, the JVM firstly ~~creates~~ loads, the parent class and then loads child class.

Error :-

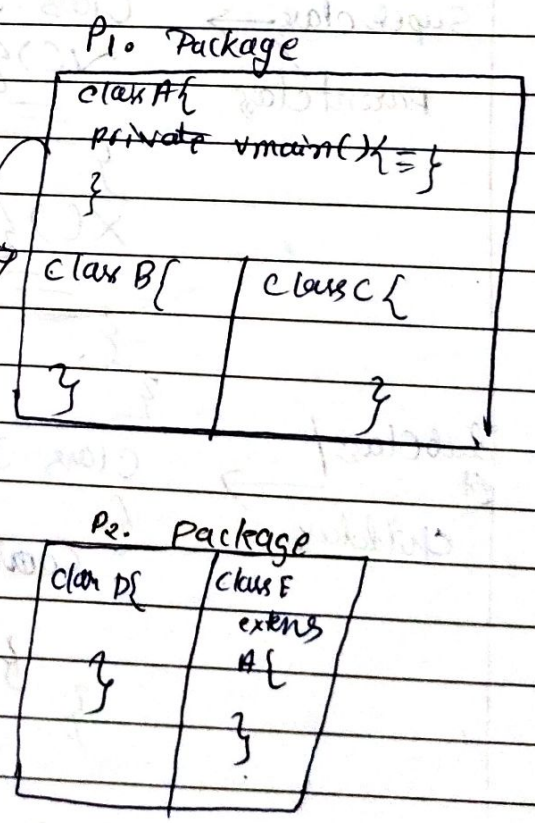
"Symbol : method receivingCall()
location : variable j7 of type J7."

"when we doesn't extends the j5 class in J7, and try to access j5 methods, than Compiler generate error."

*Default is public for the same package.

* private is accesible in within same class.

* protected member is accesible outside the package but in class child only.



Default and protected is public for the same package.

PAGE NO.:

(i) private :-
private member is only accessible within the same class.

(ii) default :-
default members are accessible in all classes of same package.
(default member is public for the same package.)

(iii) Protected :-
protected members are accessible in class classes of same package as well as child class of another package.

(iv) Public :-
public member is accessible anywhere.

"Inheritance"

PAGE NO.:

```
Class Area{
private int l;
private int w;

public void setData(int l, int w){
        this.l = l;
        this.w = w;
    }
public int getArea(){
    return l*w;
  }.
}
Class Volume extends Area{
    private h;
p v. setData(int l, int w, int h){
    setData(l, w); // this.setData(l,w);
  }
p int getVolume(){
  return getArea() * h;
  }.

Class Main{
p s vm(string args[]){

Volume v = new Volume();
 v.setData(2,4,7);
};
```

the execution order of a constructor is, the order of inheritance

PAGE NO.:

* when we create the object of class child, first it will execute the parent constructor and then child class constructor.

* If there are multiple constructor in parent class, Jvm calls default constructor by default.

If we want to inherit the Constructor, use super() keyword.

```
Class A{
public A(int x){
   S.o.p("A-.");
  }
}

Class B extends A{
   public B(){
    super(143);          → call to super() must be your first statement in Constructor.
    S.o.p("B-Default");
  }
}

class main{
 PSVM(S.. args..)
    new B();
  }
}
```

↳ अगर parent class में default constructor नहीं है, तो child class में हर constructor में super() बनाना पड़ेगा !, "

PAGE NO.:

Method Overriding :-

Types :-

⇒ Single Inheritance

⇒ Multilevel Inheritance

⇒ Hyeachical Inheritance

Method Overriding :- → why
→ what
→ How.

Why :- To change the implementation of parent class.

what :-
when we redefine parent class method in a child class with the same signature, than child class override the parent class method.

1) In case of method overeriding parameter must be same

In case of method overloading return type doesn't matter but in method overriding return type ~~was~~ matter

PAGE NO.:

3 A parent class reference variable can easily hold child class object

```
A obj = new B();
```

```
Class A{
public void wish(){
    So.p("Gm");
    }
}
class B extends A{
 p. v. wish(){
    s.o.p("GN");
    }
}
p. v. m2(){
s.o.p("B");
Clas main
    P.S.V.M(String args[]){
}
```

```
p.v. m1(){
s.o.p("A--");
}
```

① A obj = new A();
obj.wish();

② B obj = new B();
obj.wish();

③ A obj = new B();
obj.wish();
obj.m1();

[only calls parent class methods.]

Runtime binding
parent class reference, object of child class.

PAGE NO.:

- **Upcasting :-** → Directly support by Java

By default Java stores...

```
A obj = new B();
```

- **Down casting :-**

```
B obj = new A();
```

It will generate the error. Bcoz Down casting is not directly supported by Java

In Java, ~~Es~~ downcasting ~~कर सकते पर~~ उसी object को जो पहले upcast हुआ है,

```
A obj = new B();
B ref = (B) obj;
```

convert ← downcast

obj is ref

```
int x = 20;
double x = 20.5;
String s = "Hello";
```

* object is a parent of all classes
thats why we create object of object class.

```
*  d;
```

*when we don't know the type of d. then.

```
object x = d;   ← upcasting.
String s1 = (String) x;   ← downcasting.
```

"d"

• Overriding method :- जो override कर रही, over child method.

overridden method :- जो override हो चुका है, parent method.

A obj = new B() ; // upcasting.

B obj2 = new A() ; // downCasting.

---

① case :- 1

```
A obj = new A();
obj.m1(); ✓
obj.m2(); ✗
obj.m3(); ✗
```

② case : 2

```
B obj = new B();
obj.m1(); ✓
obj.m2(); ✓
obj.m3(); ✗
```

③ case :- 3.

```
C obj = new C();
obj.m1(); ✓
obj.m2(); ✓
obj.m3(); ✓
```

④ case :- 4

```
A obj = new B();
obj.m1(); ✓
obj.m2(); ✗
```

⑤ case :- 5.

```
B obj = new C();
obj.m1(); ✓
obj.m2(); // ✓
obj.m3(); ✗
```

```
class A {
    p.v m1() {
    S.o.p("A-m1()");
    }
}
class B extends A{
    p.v.m2(){
    So.p("B-m2()");
    }
}
class C extends B{
    p.v. m3(){
    S.o.p("C-m3()");
    }
```

"Rules of Overriding"

PAGE NO.:

(1) while overriding, we can change the access modifier.

(2) we cannot override the private method.

1) Private
2) <default>
3) protected
4) Public

↑ weaker access privilege.

↓ Higher acess privilege.

(3) we can change the access modifier but; the.

(4) Condition :- the access priviledge should higher or equal to the parent class.

case :- • default ✓
↓ override in.
public, protected.

• protected ✓
↓ overriding
public.

case :- Public ✗.
↓ ← override error:
protected

error :- attempting to assign weaker acces priviledge.

4) Can't change the return type while overriding

Primitive & void :-
byte, short, int, long, float, double, char, boolean, void.

child class का return type ~~of~~ change कर सकते है।

PAGE NO.:

Special Case :-

in Case of Co-variant, we can change, the return type.

```
class A{
public A m1(){
System.out.println("A-m1()");
    A obj = new A();
    return obj;
  }
}
class B etends A {
public B m1(){
S.O.pln("B-m1()");
  B obj = new B();
  return obj;
 }
}
class main{
P S V m(String args[]){
 A obj = new B();
 obj.m1();
}
```

```
class A{
public Object m1(){
   return 100;
}
}
class B extends A{
 P. String m1(){ return Hello...";
```

```
class main{
P.S.V.m()
A obj = new B();
S.O.P (obj.m1());
}
```

overriding → object related
Instance method ✓
Static method ✗.

PAGE NO.:

ये हुआ overload, न की override.

```
class A{
Public void m1(Object S){
    S.o.pln("A-m1()");
  }
}
class B{
Public void m1(String S){
    S.o.pln("B-m1()")
}
class main(){
P.S.v.m(String args[])
A obj = new B();
S.o.p obj.("Hello...");
}
```

output

A-m1()";

Static member also inherited.

The static member/method can be are the hidden but not overridden in java.

⇒ cannot override the private and static method.

In java, we can overload the main method.

Abstract Class and Abstract method.

Abstract class :-
To achieve abstraction, we use Abstract class.

Concrete class → Concrete implementation

- Concrete class :-

A java class where all methods are defined.

Abstract class → Partial implementation

- Abstract class :-

in which one method is declared, and other are defined.

- use abstract keyword to declare the abstract method.

class having atleast one abstract method is called abstract class

Interface → fully Abstraction.

अगर class के अंदर Abstract method है तो class भी Abstract होना चाहिए, और यदि class Abstract है तो जरूरी नहीं के Abstract method हो,

We cannot create the object of Abstract class.

Partial implementation is achieved through Abstract Class.

Abstract → class, method.

Abstract class can we use as parent class; It is the blueprint for the child class. (Abstract class).

when we inherit any abstract method, we have to override (cover) define that method. ~~to~~ in child class.

```
abstract class Test{
    pv m1(){
        ≡
    }
    abstract pv m2();
}
```

```
class child extends Test{
    pv m2(){
        ≡
    }
}
child c = new child();
c.m1();
c.m2();
Test obj = new child();
obj.m1();
```

* Abstract class is basically the blueprint of child class.
* abstract class is focus on what to do not How to do.

PAGE NO:

```
p v break(){
    S.O.P("Disk Brak In Reare");
}
}
class Tata extends vehicle{
  p.v puc(){
    S.O.P("...");
  }
  p v Insurance(){
    S.O.P("&&&");
  }
  p v break(){
  S.O.P("break");
  }
class TestMain{
P.S S.V M(String args[]){
  vehicle v = null;

// V = new TVS();
// V = new TATA();
v.B Insurance();
```

Why it is called the dynamic binding. It will resolve at the runtime.

PAGE NO.:

If we you want to achieve 100% abstraction, you can use interface.

Interface is the backbone of java.

"Interface"

final :-

- final is a keyword,
- final means the variable cannot be changed once it is initialized.
- So, all variables in an interface are constants.

when we declare a variable inside an interface, it is by default.

In Java,

Public static final

Example :- interface MyInterface{

int VALUE = 100; // internally ; int value
Public static final ~~value = 100~~;

This is similar to

interface MyInterface{
public static final int VALUE = 100;

public → Accessible everywhere

static → Belongs to the interface, not to objects

final → You cannot modify VALUE later

PAGE NO.:

Static → class, block
Data member, method.

Abstract
can be method, can be class.

final → class → Block, Dont inheritance
It can be
variable, method
Dont Block overriding

- we cannot override the private and static member and final.
- if we have initialize the final variable than, it cannot be change.

```
Final int x = 404;
```

- we cannot inherit final class.

★ singleton class — Immutable

```
Class A{
Final public void m1(){
=
-}
class B extends A{
X P V M m1(){
}
}
```

// m1 in B cannot be override in A

// variable X might already have been assigned

"Valid / Invalid Signature"

1) public void m1(); ✗
2) abstract public void m1(); ✓
3) static p.v m1() ; ✗
4) static abstract p v m1(); { } ✗
5) static abstract p v m1(); { } ✗
6) abstract .final p v m1(); { } ✗
7) abstract final p v m1() ; ✗
   we cannot use static and final with abstract.
8) static final p v m1(); { = } ✓

"Interface in Java"

Why what how.

why :- Providing the guide line or giving proposal to maintain the standard.

what :- Interface is the Complete Collection of abstract method.

:- Collection of abstract methods
[It only contains method declarations, not implementations]

```
interface I{
  void m1();
  void m2();
}
```

// • Methods are By default public abstract

// • Java automatically treats it as public abstract

PAGE NO.:

Rules :-

(1) Methods :-
- By default :- public abstract

(2) Variables :-
By default :- public static final (constants)

(3) Objects :-
- you cannot create an object of an interface.
- But you can create a reference of it.

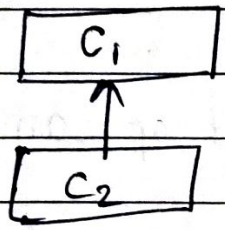
(4) Multiple Inheritance
- A class cannot extend multiple classes in java.
- But a class can implement multiple interfaces, So multiple inheritance is possible through interfaces.

- No object is form for interface.
- Interface inherited by a class.

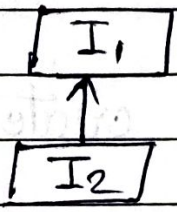
```
interface I {
void m1();
void m2();
}
class child implements I {
public void m1(){=}
public void m2(){=}
}
```

① parent p = new child();

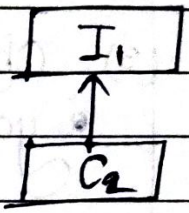
② I obj = new child();
obj.mI();
obj.m2();

① 

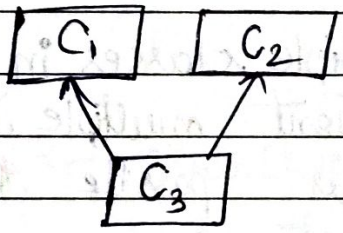
C2 extends C1

(2) 

I2 extends I1

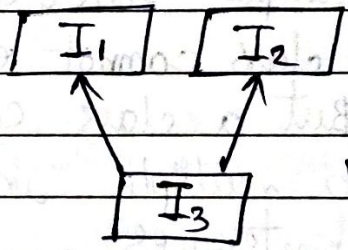
③ 

C1 implements I1

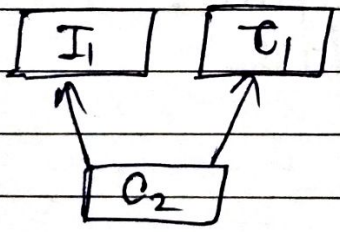
(4) 

X
C3 extends C1, C2.

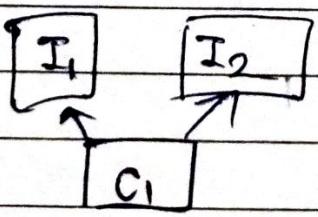
A class cannot extends multiple classes.

(5) 

✓
I3 extends I1, I2

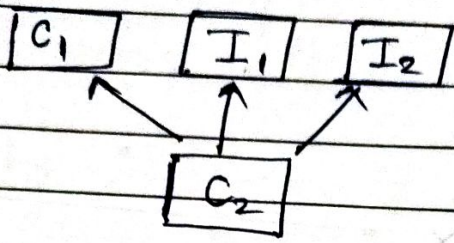
(6) 

C2 implements I1 extends C1 X
C2 extends C1 implements I1 ✓

(7) 

C1 implements I1, I2

(8)

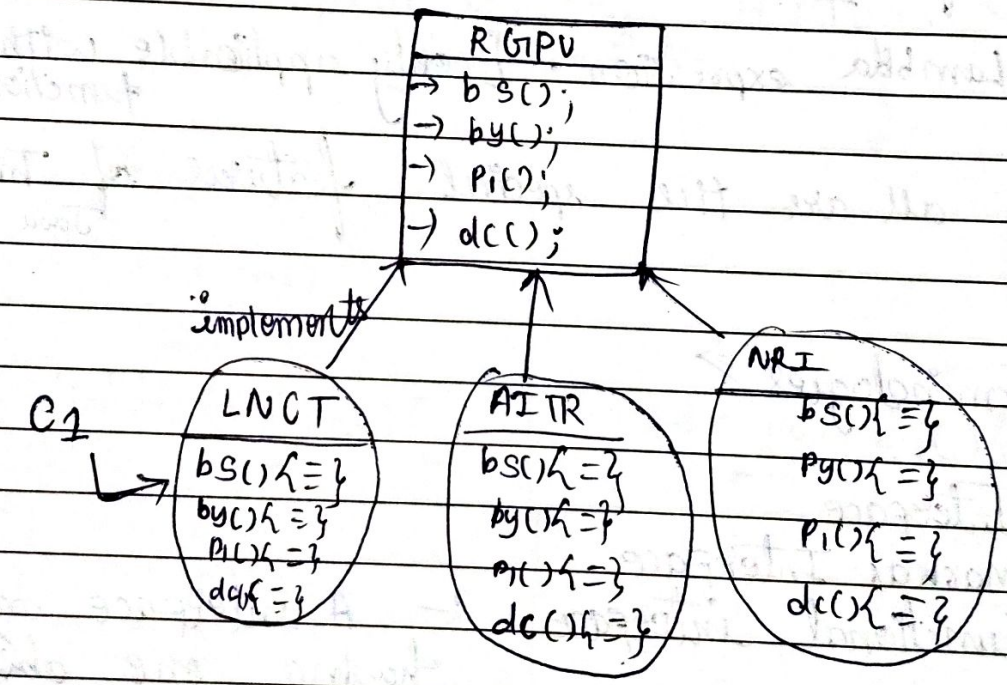


C_2 extends C_1 implements I_1, I_2

In interface :-

- No instance variable in Interface
- By Default public abstract and final.

Suppose :-



Interface List{
P V add();{=}
}

class AL{
add(){
=
}

class LL{
add(){
=
}

class vector{
add(){
=
}

(1) Multiple Inheritance
(2) No instance variable
(3) Public method.

Interface [Java 1.8].

- introduce a new termonology "functional Interface".
- Static method.
- Defined method.
  +
- Lambda expression. [only applicable with functional interface]

They all are the special features of Interface in Java 1.8

Terminologies ⇒

1. Interface
2. Markar Interface
3. Functional interface :- A interface only having one abstract method. and can have non-abstract method.

```
# Interface I{
    void m();
}
```

@functional Interface → annotation metadata.
↳ To know the JVM that this interface is functional interface and have only one abstract method.

2. marker Interface :—
→ an empty interface or blank interface.

→ marker interface is used to give the instruction to JVM. that class inherited that marker interface. It is used to mark (tag) a class so JVM or frameworks can treat it specially.
Examples :— • Serializable, cloneable, Remote, RandomAccess

Define method :—

• Default → is not access modifier X.
it is the default define by the programmer, it is a default implementation.

```
interface{
    void m1();
    void m2();
    default void m3(){
        =
        =
    }
```

we can override it

```
    static void m3(){
        =
        =
        =
    }
```

but it is not overridden

Difference b/w Abstract method / Interface

Single inheritance → Abstract method.

~~Interface~~ inheritance → Interface
multiple.

```
interface I1{
int add(int x, int y);
int multi(int x, int y);
}
class Test implements I1{
public int add(int x, int y){
    return x+b;
}
```

Anonymous Inner Class. ⇒

- An anonymous inner class is a type of inner class without name.
- It is declared and instantiated at the same time.

⇒

```
interface I1{
int add(int x, int y);
int multi(int x, int y);
}
I1 obj = new I1(){
public int add(int x, int y){
  return x+y;
}
public int multi(int x, int y){
  return x*y;
}
};
obj.add(20,10);
```

Internally it creates a class

```
class ___ implements I1
{
}
```

behind, the Java creates child class of I1 interface, So, we can make the object of I1. interface.

```
class Testmain{
    p s v m(String args[]){
    I1 obj = new Test();
    S.o.p (obj.add(20,10);
    S.o.p( obj.multi( 20, 10);
    }
}.
```

Inner class
I1.class
Testmain1.class
Testmain$1.class

~~Lambda Expression~~ :-
(interface must be ~~Functional~~ Interface)

Marker Interface :- A marker interface is a interface with no methods and no fields.
it is used to mark(tag) a class so JVM or frameworks can treat it specially.

ex:- Serializable, Cloneable, Remote, RandomAccess.

why do we need marker Interface ?
marker interface give metadata about a class.

- Serializable :- if a class implements Serializable,
  JVM knows :
  - "This object can be saved to a file or transferred over a network."
- Cloneable :- if a class implements Cloneable,
  JVM knows :-
  "This class allows cloning using the clone() method".

Concepts ⇒

PAGE NO.:

classloader's.

① Bootstrap classloader ⇒ loads utility classes

② extension CL ⇒ Java.X. package . classes

③ Application CL ⇒

new B();

static block of A..
static block of B
initializer of A
Constructor of A
initializer of B.
Constructor of B.

```
class A{
public{ // initializer block.
    }
public A(){
    }
.static (){
    }
class B extends A
{ // initializer block
}
public B(){
    }
```

```
static(){
    }
Class Test main{
  PSVM(S...a9[]){
    new B();
  }
static
```

Lambd expresion is a consaise way to represent an anymonaus. function (a function without aname), lamiba expression provide a more compact and expresive. syntax for implementing functional interfaces, which are interfaces with a single abstract method.

PAGE NO.:

Lambda Expression :-

Lambda expression only applicable on functional interface. it given functional programming

Lamba Ex. Syntax.

(parameter) → body (expression).

Characteristics :-

① • Optional type declaration.

② • optional parenthesis around the parameter.
③ • optional curly braces
④ • optional return keyword.

#

```
interface Greetings {
    void wish();
}
class TestMain{
PSV...
Greeting obj1 = ()→System.out.println("GM ...");
Greetings obj2 = ()→System.out.println("GN ...");
Greetings obj3 = ()→System.out.println("GF ...");
    obj1.wish();
    obj2.wish();
    obj3.wish();
```

PAGE NO.:

Top level class can be public or non-public.

In java we can create a static class but the class must be the childclass.

```
class A{
  static class B{
    public void wish(){
      S.o.p("GM---");
    }
  }
}

class Test{
  P S v M(S.. args[]).

    A.B ref = new A.B();
     ref.wish();
     A.B.wish();   If the method is static =>
```

Abstract Method. vs. Interface.

Method Type :

Abstract class :-

- can have abstract ~~class~~ methods
- can have concrete methods (fully implemented).

Interface

Can have abstract methods (implicity public and abstract).

from java 1.8 it can have concrete (default methods) + static

PAGE NO.:

*Variable

A.C.

- can have instance variable and static variable.
- variables can have any access modifier (private, default, protected, public)

Interface

- Can only have public. static Final (constants)
- No instance variables.

*Inheritance.

A.C.

- Supports single inheritance only (class can extend only one abstract class).

- Interface
- cannot have it supports multiple inheritance

*Constructor =

A.C.

- Can have Constructor

Interface

cannot have Constructor

PAGE NO.:

Access modifier

- A.C

methods combe (public, private, protected, default)

- Interface

All methods are public by default. (except private methods java 8).

when to use

Abstract class

when you want to use share code (static + behavior) among related classes.

Interface

~~when you want to use.~~

use when classes share only a contract (behaviour) but not implementation.