Vikash Polytechnic, Bargarh

Vikash Polytechnic
Campus: Vikash Knowledge Hub, Barahaguda Canal Chowk, NH6
PO/DIST: Bargarh-768028, Odisha

Lecture Note
on
Object Oriented Methodology

Diploma 2ND Year (3RD Sem)

BRANCH-CSE

Prepared by: - Ms. Monali Patel
Senior Lecture

Computer Science & Engineering

Programming Language :-

Language :-

→ Language is a mode of comm i.e used to Share ideas, opinions with each other.

→ eg :- If we want to teach someone, we need a language i.e understandable by both communicators.

What is Programing Language? -

→ A PL is a computer language i.e used by programmers to communicate with computers.

eg :- C, c++, Java, Python etc.

Types of PL :-

a) Low Level Language

b) High Level language

c) Middle Level language

a) Low Level language :-

Low LLL is machine dependent PL.

→ The Processor run low-level language directly without need of compiler or interpreter. (So it runs so fast.)

* i) <u>ML</u> (Machine Language):-
. 4 machine code on object code.

→ Easier to Read becoz ~~it interpreted coad~~
it normally displayed in binary on hexadecimal form.

* ii) <u>Assembly Language</u>:-

Set of instructions in a symbolic & human-understandable form.

→ Assembler to convert AL to ML.

→ Less memory less execution time to execute program <u>eg</u>:- mov 1
add 1

b) <u>HLL</u> :→

Designed for developing user-friendly sml Program & websited.

→ Require compiler or interpreter to translate the Program into ML.

<u>eg</u> : Java, C, C++, C#, etc

i) <u>Procedural oriented</u> PL :-

↳ ~~Derived form~~ Procedure call concept
& It divides a program into small Procedures called routines or fun".

<u>eg</u>:- IDE, microsoft visual studio.

C, FORTAN, Basic.

ii) OOP

↳ Based upon the objecte.
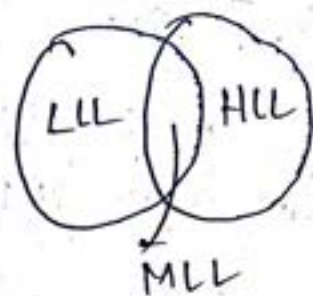
→ Programs are divided into small parts called objects

eg. Java, Python

c) iii) Natural language :-

Part of human languages such as English, Russian, German etc.

c) MLL : →

lies bet^n LLL & HLL

, C, C++.   Megha dear

MLL

Description of compiler & Interpreter

compiler → computers program

compiler : → converts the HLL to MLL program written

→ It translates the whole program at a time.

eg Java, C.

Ⓐ **Interpreter :→**

converts instructions line by line written in HLL to MIL

→ It translates the program line one by one.

✳ imp Difference bet^n compiler & Interpreter

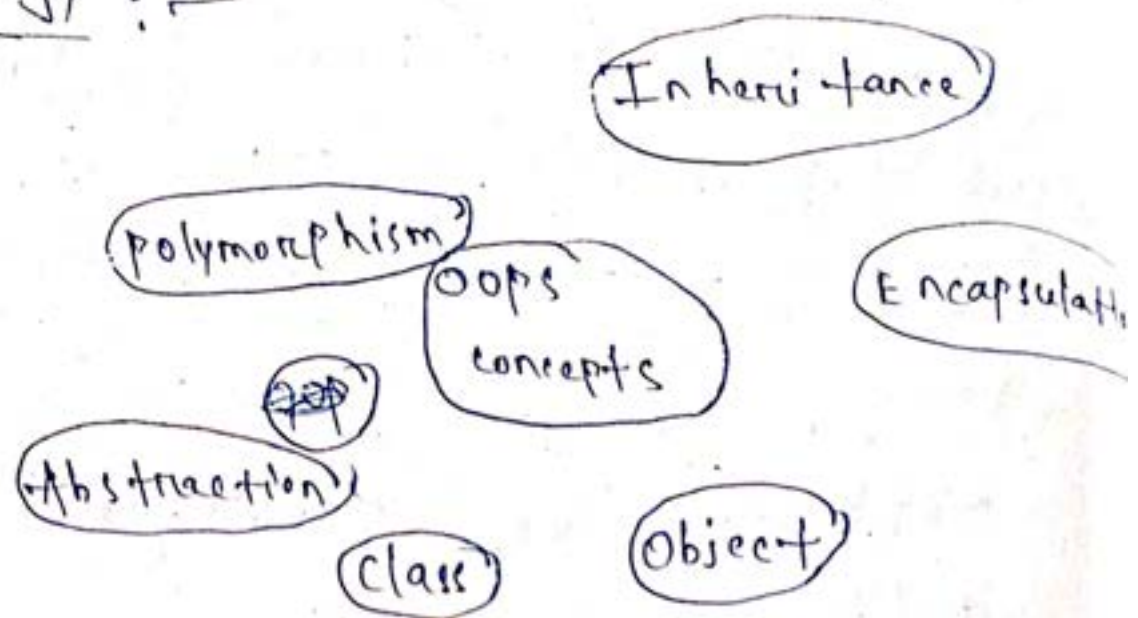| compiler	Interpreter
→ Translate the entire program at a time	→ Translate program line by line.
→ more memory required	→ Less memory
→ Execution time is less.	→ Execution time is more
→ If error detected nothing is executed	→ If error detected it stops & do not go to next line
→ c, c++ etc	→ Perl, Python, etc.

OOP

→ It is a methodology or paradigm to design a program using class & object.

Terminology :-

(Inheritance)

(Polymorphism) (OOPs concepts) (Encapsulat...)

(Abstraction) (OOP)

(Class) (Object)

(Object oriented Programming)

* Difference betⁿ Procedural & object op

OOPS concepts & Terminology   20.10.21

→ OOP is a programming paradigm (Prototype/model)
that relies on the concept of classes &
objects.

→ It is used to structure a software program
into a simple, reusable piece of code.

imp
* Features / characteristics of OOPs :→
There are   main oop concepts in Java,
These are :-

a) Object   d) Polymorphism

b) Class   e) Abstraction

c) Inheritance   f) encapsulation

a) Object:→

→ Objects are the basic runtime entities (that has state & behaviour) in an object oriented system.

→ An object can be defined as an instance of cls.

eg:→ Dogs have state(name, color, bred, hungry) & behaviour(barking, fetching, wagging tail).

b) Class:—

→ collection of objects is called class.

→ It is a logical entity

eg:→ each bicycle was built from the same set of blueprint & therefore contains the same component

c) Inheritance:—

→ When one object acquires all the properties & behaviour of Parent-object is known as inheritance.

→ It Provides code reusability. It is used to achieve runtime polymorphism.

→ eg:→ Dog, cat, cow can be derived class of Animal base class.

Some imp. terminology:—

i) Sub class: — The class that inherits properties from another class.

ii) Super class:— The class whose properties are inherited by Sub class.

iii) Reusability :—
when we want to create a new cls &
there is already a cls that includes some of
the code that we want, we can derive our
new class from the existing cls. (BY reusing
the fields & methods of the existing cls).

d) Polymorphism :— (More than one form)
→ When one task is performed by different
ways i.e known as polymorphism.

eg :→ To convience the customer differently, to
draw some thing eg. shape, or rectangle etc
→ In Java we use Method overloading &
method overriding to achieve Polymorphism.

* Dynamic Binding :—
when the type of the object is determined at run-time

e) Abstraction :→
→ Hide internal details & showing functionality is
known as abstraction.
eg :— Phone call (we don't known the internal
processing)
→ In java, we use abstract class & interface
to achieve abstraction.

f) encapsulation :—
Binding (or, wrapping) code. & data together

into a single unit

→ eg :→ A capsule is wrapped with different medicine

→ Java Bean (→All the data members are Private here,

→ Provides Security that keeps data & method safe from inadvertment change.

→ (Hiding data & method with in the object)

Benifits of oops :→

oop has great advantages over other prognaming stylee.

a) →) We can eliminate redundant code & extend the used of existing class. (Through inheritance).

(b) Helps the programmere to build Secure progra (Principle of data hiding).

c) Possible to have multiple objects to coexist without any interberence.

d) The object controle how one interacts with others, preventing other kinds of errors.

e) Appln

1) client - Server Systems (internet

2) object oriented db

3) Real time System design

4) simulation & modeling System

What is Java?

→ James gosling 1995,

Java was developed by James Grosling in 1995

→ A general purpose object-oriented language.

→ Write once Read Any where (WORA).

→ Designed fore easy web/Internet appl".

→ Platform independent PL.

→ Java contains extensive library. (2500)

imp

* Why platform independent?

Becoz of WORA structure makes Java as platform independent i.e write codes in one os fore example windows XP & execute that code in an othere platform (os) i.e MACos & linux.

Why Java

→ Simple

→ object oriented

→ Distributed

→ Interpreted

→ Robust

→ Secure

→ Portable

→ multi threaded

History Behind Java :→

→ ~~James Grosting~~ James Groling is the father of Java, co-founder of Java was vinod khosla.

→ Java was introduced in "Oak", may 20, 1995 at sum world.

Java was catagorise into 3 basic types

a) ~~Jar~~ J2SE (Java 2 Standard Edition):—

→ To develop client side stand alone appⁿ such as Thread, File Io, Swing, Applet.

b) J2ME(Java 2 Micro Eddition):—

To develop appⁿ for mobile devices such as cell phones.

c) J2EE (Java 2 Enterprise Edition):—

To develop serverside appⁿ such as Java Servelet & Java server pag

→ Java was originally generated by Sun microsystem, But now Java was purchased/ owned by oracle corporation.

How is Java different from c

→ Major difference is that c is a Structure orient & Java is an object oriented (class & object

→ Java doesn't support an explicit poi...

→ Java doesn't have preprocessor, So, we can't
use # define,

→ Java doesn't include structures unions &
enum data types.

→ Difference betⁿ c, c++ & Java

C	c++	Java
1) Procedural lang.	→ object oriented PL	→ pure OOPL
2) Static PL	→ Static PL	→ Dynamic PL (operation done on run time also).
3) code is executed directly	→ directly	→ code is execu by JVM
4) platform dependent	→ platform dependent	→ Platform independent
5) uses only compiler	→ compiler	→ Both compiler & interpreter (Interpreted language).
6) generate .exe & .bax, files	→ .exe	→ .class file
7) 32 keywords	→ 60	→ 52
8) .c	→ .cpp	→ .java
a) support pointer	→ also	→ doesn't

→ So doesn't
   support constructor
   & destructore

→ Both

→ only consto

→ malloc(), calloc(),
   free(), realloc().

→ new &
   delete
   operatore

→ garbage
   collectore.

→ An array, should
   be declared with
   size
   a[10];

→ Same

→ An array
   can be
   declared
   without
   declaring
   a[] the size

Execution Model of Java
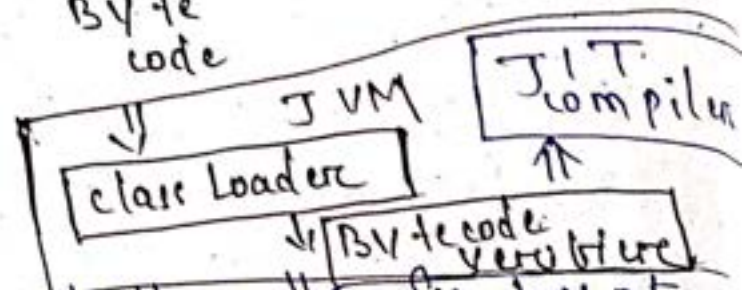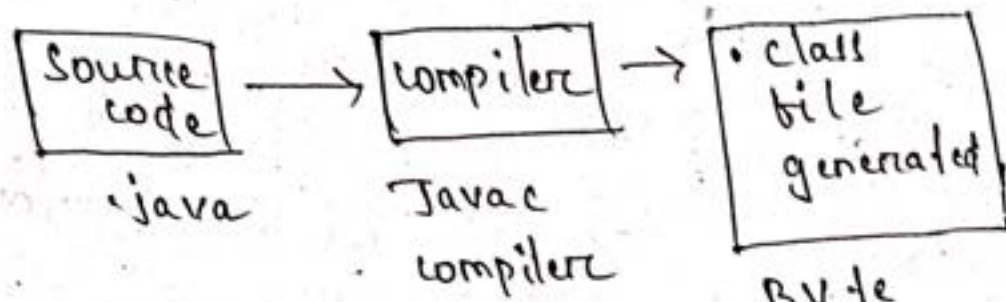
Prerequisits By start writing Java

program

1) Setup youre Work Environment.
2) Any Sort of Notepad- to write java progr
3) Make Sure you have The JDK installed
4)   Tool useful fore

developping & testing programs Written in Java
& running on the java platform.

Execution moodel of Java

2 Steps
a) compilation → Javac compilere
b) Execution → JVM

```
┌─────────┐      ┌──────────┐     ┌──────────────┐
│ Source  │ ───→ │ compiler │ ──→ │ • class      │
│  code   │      └──────────┘     │   file       │
└─────────┘                       │   generated  │
  •java          Javac           └──────────────┘
                compiler
                                  Byte
                                  code
```

Byte code

JVM   JIT compiler

class Loader

Bytecode verifier

→ Javac compiler checks whether the syntax of the program is correct or not.

 → If it successfull we get .cls file
                                    (Byte code)

Byte code :⇒ Machine independent code
      & not   (we can run the code in any
  os) readable

Run :—

→ Gives the .class file to Java class loader

   Javac clsname of mainmethod⟩

→ class loader loads all the classes which is required for complete info of java program

→ Byte code verifier (Run time error)

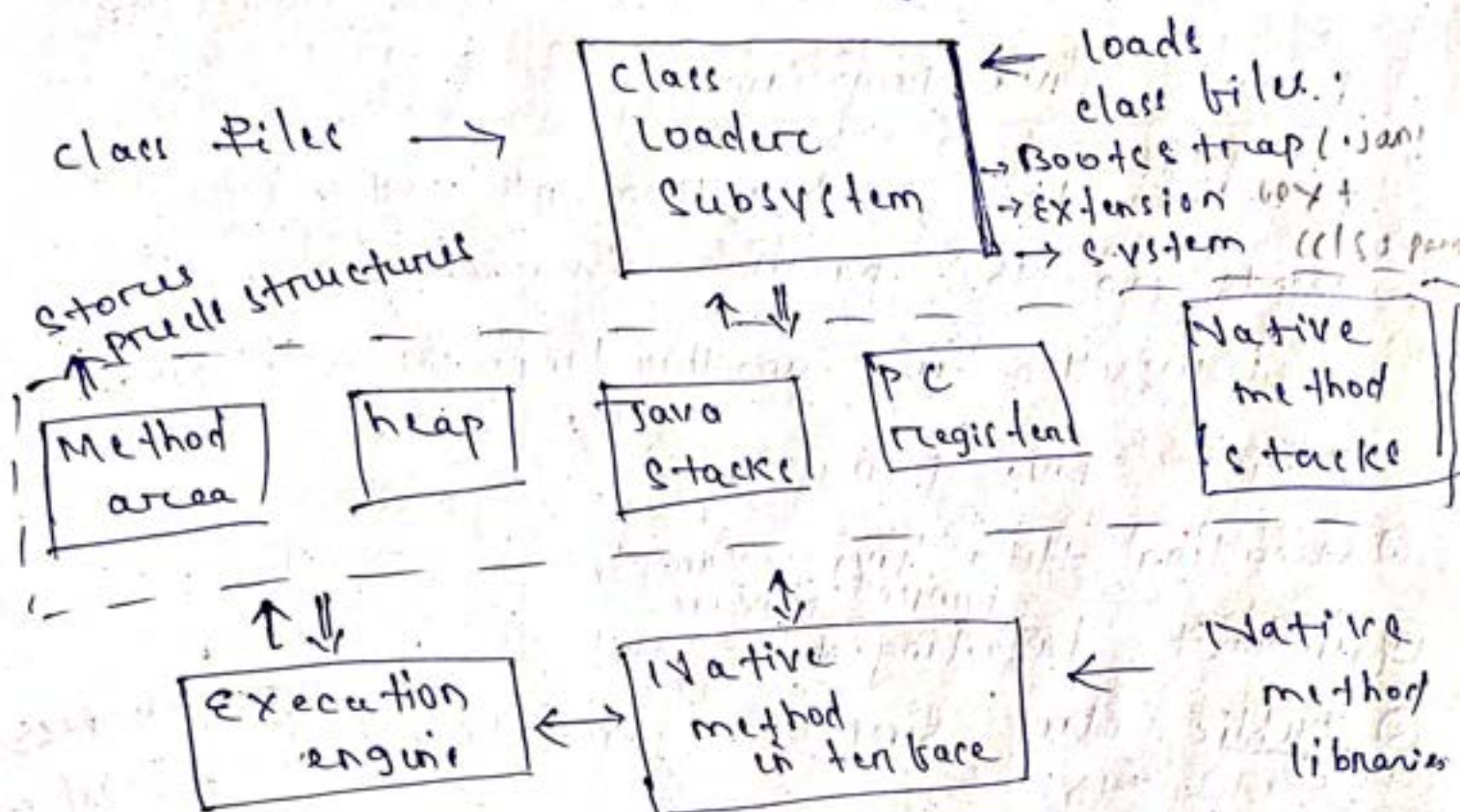→ JIT (Just In time) compiler converts byte code to machine code.

JVM (Java virtual Machine)

→ JVM converts Java Byte code into machine lang.

⊛ **Architecture of JVM (Java virtual Machine)**

- Architecture of JVM contains classloader, memory area, execution engine etc.

class files →

```
                    ┌─────────────┐      ← loads
                    │   Class     │        class files.
                    │   Loader    │      → Bootcstrap (.jar
                    │  Subsystem  │      → Extension boot
                    └─────────────┘      → System class path
stores
↑ preclass structures
┌────────┐   ┌──────┐   ┌────────┐   ┌──────────┐   ┌─────────┐
│ Method │   │ heap │   │ Java   │   │   PC     │   │ Native  │
│ area   │   │      │   │ Stack  │   │ register │   │ method  │
└────────┘   └──────┘   └────────┘   └──────────┘   │ stack   │
                                                     └─────────┘
↑                          ↑
┌────────────┐   ┌────────────┐         ← Native
│ Execution  │   │  Native    │           method
│  engine    │ ← │  method    │ ←         libraries
└────────────┘   │ interface  │
                 └────────────┘
```

class/Method Area :—

Store all Preclass Structures.

eg :— run-time const. pool

Heap stores the object

Java stacks :— Methods or variables

program counter register : →

     It stores the address of the currently executing instruction by JVM

Native Method stacks   non java code (C or c++)
    Native methods (JNI)
  Java   c++
  m() ⟷   → Bridge (JNI)
            { }
Execution Engine :-
  virtual processor, interpreter, compiler

  Native method interface ⟷ Java native libre

First Java program :→

  holds
→ store the instruction of native code
depends on the native library.
→ It is written is another lang. instead of java.

    First Java program

① creating Hello java example -
              project   package
②② import java.lang.*;→ All cls
                      ①
③ .public class simple   /* Hello */   #includecst
    {   & void Pub
    ④ public static void main (String args[]) {
      {
        System.out.println ("Hello");
                                              }
      }
    }   print(f(-.
  }

① Package Declaration :→
  ⊘ package package name
  Built in   built in or user defined
② import statement
  import built in / user defined library

3> clss declaration

    public class → any package can be accuse

    Main Method

→ class : → Key word is used to declare a class in Java

public : → Access Method i.e useable to all.

static : → There is no need to create object
to create object to invoke the static method.

→ The main method is executed by JVM, so it
doesn't required to create object to invoke
main()

void → Return type of the method, means it
doesn't return any value.

main :— Startup of the program

n) String args[] : → connect line argument.

y System.out.println :— print statement

D·l - 25·10·21

variable → name of memory location whole value
varies, that may ore may not change.

Datatype : → specifies to the compiler
which kind of it is.

"+" is used to join 2: string (concatenate

use ; 3rd >>

Data type :⇒

a) Premitive data type

b) Object / reference data type. / Non premitive

a) Premitive Data type

→ Predefined by the programing language called as the keyword brough by the compiler.

→ There are 8 Premietive data type

i) Byte :→

→ Used to save space, may be inplace of integer becoz a byte is 4 time smallere than an integer.

→ 8 bit signed (may be -ve or +ve)

→ -128 to 127   eg :- byte a = 10;

ii) Short

→ 16 bit signed 2s complement integere

→ -32768 to +32767   eg short a = 324;
                           short n = -5000;

→ Save space(2 times smaller than integer)

iii) Integere

→ 32 bit signed 2's compliment integere

→ -2147483648 to 2147483647

→ Default value is zero   → eg int a = 80;

iv) **Long**

- 64 bit signed 2c complement integers
→ -9223372036854775808 to 9223372036874775807
→ used in wider range than integers
→ Default value is OL
→ x:- 9567890L   eg long m = 10000L;

v) **Float**

→ It is a single prusigion, 32 bit IEEE 754 floating point.

→ Float mainly used to save memory in large array of floating point no.

→ default value is 0.0f

→ Never used for prosized value such as currency

→ eg   x = 259.5f   float F1 = 234.5f;

vi) **Double**

→ Double prusigion IEEE 754 floating point.

→ 0.0D   double a = 12.3;

vii) **Boolean** :→

→ One bit information datatype used for logical aspects.

→ 2 value True/false   boolean a = True;

→ Default value is false.

→ eg   boolean x = True

                       else ; 3rd))

viii) <u>char</u>

→ Single 16 bit unicode char.

→ \u0000 to \uFFFF   char letter= 'A'

→ 'char' is used to store any character.

WAP to add two no.

```
class Add
{
    public static void main (String args[])
    {
        int a= 10, int b= 20;
        int c= a+b;
        System.out.println(" sum is = " + c);
    }
}
```

WAP to Print a character / string

```
public class A
{
    public static void main (String args[])
    {
        char a1 = 'a';
        char a2 = 'b';
        System.out.println(" char 1:" + a1);
        System.out.println(" char2:" + a2);
    }
}
```

b) Non Primitive data type

c) user defined data type

u types of primitive data type

i) Array :—

→ Used to store data in consecutive manner

ii) String :—

→ used to store consecutive characters.

string abc = " Hello ";

iii) class

iv) Inter face

Variable : →

A name of the reserved area allocated in memory. (Name of the memory location)

in + data = 50;

↳ variable

3 types of variable

a) Local variable → C inside the body of the method()

b) Instance varciable / Non-static variable → inside the class but outside the body of the method)

c) static variable

↓

declared as static

```
Public class A
{
        Static int m = 100; //→ Static Variable
        void method()
        {
                int n = 90; // local Variable
                int m = 20; int s = m + n;
        }
        public static void main (String args[]
        {
                int data = 50; // instance Var
        }       int int b = m + data;
}
```

Operators
─────────

Symbol i.e used to perform operation

→ unary operators

→ Arithmatic operators

→ shift operators

→ Relational operators

→ Bitwise operators

→ Logical operators

→ Ternary operators

→ Assignment operators

a) unary operators

→ Java unary operators requires only one operand:

operation : →

a) postfix (exp++ / exp--)
b) prefix (++ exp / exp--)

eg :→ int a = 10;

$(++ a);$ → 11 $a++ = a+1$

$(@++)$ → 11 (12) $a++$
                        $=$
$(--a)$ → 11 $a++$
                        $=$
$(a--)$ → 11 (10) o

pt WAP for unary operator

class unary
{

          a = 10 (11)
          a++ → 10
          a++ (11)

    public static void main (String args[])
    {
        int a = 10;
        System.out.println(a++); → 10 (11)
        System.out.println(++a); → 12
                          (a--); → 12 (11)
                          (--a); → 10

+, -, *, /, %

⇒ Java Arithmatic operator Example

public class operatores
{
  public static void main (string args[])
  {
    int a = 10, b = 20;
    7u System.out.println (a + b);
    System.out.println (a - b);
    System.out.println (a * b);
    (a / b);
    (a % b) → 0

    Reminder
"

"

By using Scanner cls : → ← import Scanner cls
    ↑ user input

import java.util. Scanner;
    ↳ collection of frame work, legacy
collection clses, event model, date & time
facilities(string tokenizen, Random no. generat
                    on

public class A
{
  p svm (String args[])
  {
    int a, int b, int c;
    Scanner sc = new Scanner (System.in); ← construct Scanner cls object
    System.out.println(" Enter 1st no.");
    a = sc. nextint ( ); ← define variable to
    System.out.println (" Enter 2nd no.";
    b = sc. nextint ( ); Receive u/p

```
c = a + b;
System.out.println(" c);
}
}
```

for double

nextDouble() → for double   nextBoolean() ← bodean
nextFloat() → float   nextShort() → short
nextLine() → string
nextlong() → long

Q: WAP to print a person's Age, Name &
salary by using Scanner cls.

c) Logical operators :-
    When we want to form compound by
combining two or more Relation.
→ && , !! , !.
→ Logical expression defere a value of true
or false.

op1	op2	oper1 !! ope2	&&
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	F

d) Relational operators
    → It is used in the case of comparision
                            3) else + 3rd 99

$==, \ !=, \ >, <, <=, >=$ T/F

↳ True / false
int a = 5, b = 6;
System.out.println(a > b);

Logical or / & & operations   (10)   10

int a = 10, b = 6, c = 15;

System.out.println(a > b && a > c);

e) Shift operatore :-
<<, >> → Right shift
→ left shift   Right shift
public class Shift {
public static void main (String args[])

int a = 8; → 1000
int b = a << 2 ; ⇒ extra 0's
int b = a >> 2 → cut the Right
value

32 16 8 4 2 1
8 ⇒ 1 0 0 0 00 0 ⇒ 32

1 0 0 0 0 0 → 3 2
↳ add extra 2 °0's

8 → 1 0 0 0 :
↑0 0 0 00
>> 2 ⇒ 10.

2.5
→ a = 1 1 0 0 1

a << ⇒ 2   1 1 0 0 1 0 0 (add 2 0's)

>> 2 ⇒ 1 1 0 (cut 0)

#) Bitwise operatore
→ Execute single bit of their operands.

| $|$ → OR (Bitwise)
& → Bitwise AND
&= Bit wise AND Assignment
|= Bit wise OR "
∧ → XOR (Exculsive OR)
<< → left shift
>> → Right shift
~ → one's complement

int a = 5 ∴ int b = 7

a   b
0   0111
0 101 ∧

a		b
0		0
1		1
0		1
1		1

XOR
0 ─ 0 = 0
1 ─ 1 = 0

&	∧	~→single
0 8	0 8 0	~a
1	1 9 1	
0	1 2 1	0
5.	1 0 0	1
	7	1
	/2	0

0 111
0 → 0 = 0
1 - 1 = 0

0 1 0 1
+   1
─────────
0 1 1 0

6
(-ve)

int a = 5, b = 7;
system.out.println (a & b);
(a | b);
a ∧ b);
(~ a ∧ b);

1 0 0 0 0 0 → 32
32 16 8 4 2 1
1 0 → 2

g} Ternary operatore

    conditional operatore

var = (cond^n) ? value 1 : value 2

           ↗ if true   ↗ if false

min = (a < b) ? a : b ;

min = 2 < 5   2

                                                a = 2

                                                b = 5

h} Assignment operatore :-

    Assign value on e te right

 on e te left

      a = 10 ;   b = 20

      a += 4 ;   a = a + 4

      b -= 4 ;   b = b - 4

                                a += 4 ; a = a + 4 ; to the opera

                                  a = a + 4 ; a += 4 ;

Type casting :- Type conversion

→ convorting one premitive datatype into
another premitive data type is called
premitive data th casting / conversion.

      int   a = 5 ;

      double b = 2.7 ;

    int c = a + b ;   5 + 2

    double c = 5.0 + 2.7

WAP to calculate division
WAP to find max^m of 3 no. using
nested if else statement

```
class Test
{
    Psvm ( String args[ ])
    {
        int    a = 6, b = 5, c = 13;
        if ( a > b) (&& ( a > c)
        {
            System.out.println ss a is larg
        }
        else if ( b > a) && ( b > c)
        {
            System.out.println ss b is la
        }
        else
        {
            system.out.println ss c is grea
        }
    }
}
```

WAP to calculate division obtained by
a student   with his % mack given

```
a = 76;
if ( a > 60%;
{ 1st;
```

```
else if ( a > =50 &&
              a < 60)
{ 2nd
} else ; 3rd ))
```

9/11/21

Switch Statement :-

→ The Switch " is Java's multi way branch
→ It is a case/selection oriented statement whre
  among the alternative we have to choose a
  single alternative.
→ It is better than nested if else and if-els
→ if ladder statement.

  ← user define variable
     or const.

Syntax

```
Switch (choice/expression)    ↙ Built in or mandatory
{
  case 1 :
    S1 sequence;
    break;  ← may or may not be
  case 2 :
    // statement sequence;
    break;
    .
    .
    .
  default :
    S1 :
    .
    Sn
}
```

```
int number = 20;
Switch (number)
{
  case 10:
    System.out.println("10");
    break;
  case 20:
                    ("20");
    break;
  case 30:
                    ("30");
    break;
```

default:
  C.o.p("Not 10, 20 or 30");
}

Note :→ i) There can be one or N no. of case
         value must be literal.
         ii) The case value must be switch exp.
            type only. c it doesn't allow variable
         iii) The case value must be unique.
            In case of duplicate, it rendere
            compile time errore.
         iv). Each case statement can have
            a break statement which is
            optional. when controls reach to
            break it ~~bre~~ jumps the
            controle after switch.

Jump Statement:—

Jump Statement makes the control jump to
another section of the program.
i) Break Statement:→ Takes control out of
the loop.
ii) continue statement → Takes control the
            begining of the loop.

iii) .go to statement→ To desried line ol
            ^program

iv) Return

Class :→

→ class is a rule or structure using that structure we give real life existance/ implementation.

→ class is a template or blueprint from which objects are created.

→ Everything we wish to represent in a Java program must be encapsulated in cls.

→ cls defines the state & behaviour of the basic programing components known as objects.

→ clses creates objects & object use methods to communicate bet^n them. (All about OOP)

eg :→

class	properties	Behaviour
Human	Height	eat
	Weight	talk
	colore	sleep
	Name	

Anatomy / Syntax of cls :→

modifier class Classname
{
     ~~Access~~
     level   { datatype variablename;
              datatype method name(p1, p2)
           =
     }
}

factorcial of 'no.'

class Fact

```
{ p s v m ( String args [])
    {  int n=5, fact=1,P,
       while (n >= 1)
          {   fact = fact *n;
                 n--;
          }
       System. out. println ( fact);
    }
}
```
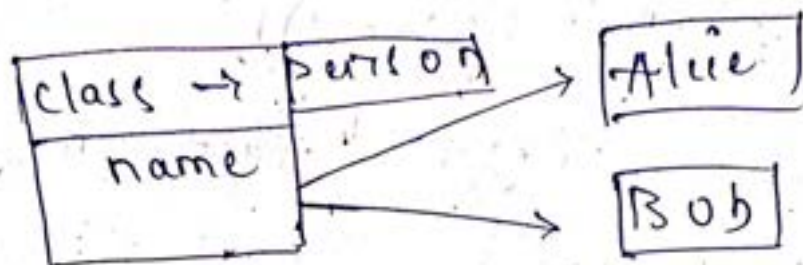
Static block, Method, variable

variable → Static priod

⅍ 2 ways

i) we can directly access

ii) with the help of cls nam

→ Static
Related to cls

class → person → Alice

name → Bob

→ indicates that the particulare member
belongs to a type itself. mainly used
to help memory management.

1 copy only (not change in cls→)

eg. name   No- No value changes is Stati
breed : Homo

applied in variable, cls, method, blocks,

nested cls

Final

↳ Static keywords can't be declared in
non Static cls.

* We can write Static in main cls
We can't not write nonstatic method in
Static method.

Jump Statement:-
y Transfere Program execution control to
a specific statement.

a) Break : ⇒
class A
{
    psvm(....)
    {
        for(int i=0; i<10; i++)
        {   if(i==5)   0
            {   break;   1
                                          2
            s.o.p(i);   3
        }   4
        s.o.p("outside");
    }
}

b) continue :→ It is used within looping
statements.
class A   1
{   psvm(String args[])   3
    {   5
        for(i=1; i<10; i++)   7
        {   if(i%2==0)   .
            {   continue;   .
            s.o.p(i);   99
} } }

Return

It is mainly used in methods in ordere to termi..
a method in bet^n & returen back to the
callere method

```
class A
{
    double add(double a, double b)
    {
        double sum = 0;
        sum = (a+b) / 2.0;
        return sum;
    }
}
```

<u>Syntax :→</u> Return
          returnvalue.;

<u>program :→</u>
```
public class A
{
    int comparue()
    {
        int x = 3;
        int y = 8;
        S.o.ipc " x = "x + \ny = " +y ");
        if (x>y)
            return x;
        else
            return y;
    }
    psvm()
    {
        A ob = neul A();
        int R = ob.comparue();
        S.o.p(+R);
    }
}
```

Loop in Java

→ Looping repeatedly execute the same set of instruction until a termination cond" is met.

→ It reduces the programming effort fore execution of similare statement fore a no. of time or infinitely time

(orc) If we want to repeat one statement multiple times, in that case we can simply use loop/ iteration statement.

→ Loops must have a starting point it may or may not have ending point.

Types

a) while loop :

→ It is used to iterate a part of the program several times.

→ If the no. of iteration is not fixed, & it is recommended to use while loop.

Syntax : ⇒

```
while (condition)
{
    code to be executed;
    increment / Decrement (may or may not)
}
```

Q) WAp to print 1 to 10

Ans:-

```
public class A
{
    public static void main (String args[])
    {
        int i = 1;
        while ( i <= 10)
        {
            System.out.println (i);
            i++;
        }
    }
}
```

Infinite loop using while

```
public class infinite
{
    public static void main (String a[])
    {
        while (true)
        {
            System.out.println ("infinite loop");
        }
    }
}
```

Flowchart
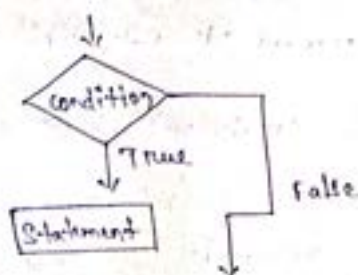
b) do-while Loop

→ It is used to iterate a part of the program several time

→ If the no. of iteration is not fixed & you have to execute the loop at least once, it is recommended to use do-while loop.

→ Java do-while loop is executed atleast once before cond" is checked after loop body

Syntax :-

```
do
{
    code to be executed;
    increment / Decrement
} while (condition)
```

Flowchart :-
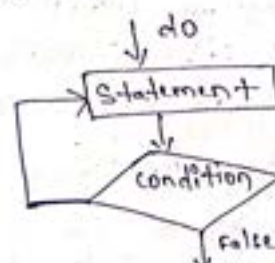
Q w/Ap to print 1-to 10 using do-while

```
public class A
{
    public static void main (String args[])
    {
        int i;
```

```
do
{
    System.out.println( i );
    i++;
} while ( i <= 10)
}
}
```

Java infinitive loop

```
do
{
    System.out.println(" infinite loop");
} while ( true)
```

c) For loop
→ The java for loop is used to iterate a part of the program several times.
→ If the no. of iteration is fixed, it is recommended to use for loop.
→ It consist of 3 parts
i) Initialization :
→ It is the initial cond⁷, which is executed once when the loop starts
→ Here, we can use an already initialized variable ore we can initialize a variable.

ii) Condition :
→ It continue execution until the condition is false.
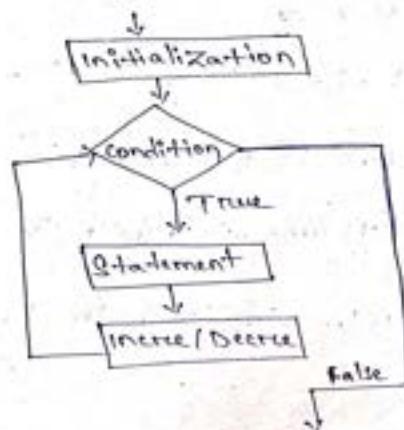→ It must return boolean value i.e either true/false.

iii) Increment/Decrement :
→ It increments ore decrements the value.

Syntax
```
for(initialization; condition; incre/decre)
{
    Statement → to be executed;
}
```

Flowchart

Q. W-AP to print 1 to 10 using

```
public class A
{
    public static void main(String args[])
    {
        for(int i=1; i<=10; i++)
        {
            System.out.println(i);
        }
    }
}
```

Pyramid

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

i = 1
*

```
for(int i=1; i<=5; i++)  ← Row
{
    for(int j=1; j<=i; j++)  ← column
    {
        System.out.println("*"); -
    }
    System.out.println();  ← newline
}
```

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

```
for(int i=1; i<=5; i++)
{
    for(j=5; j>=i; j--)
    {
        System.out.println("*");
    }
    System.out.println();
}
```

Java Infinite loop

```
for(;;)
{
    System.out.println("Infinite loop");
}
```

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

```
for(i=1; i<=5; i++)
{
    for(j=1; j>=i; j++
    {
        s.o.p(j);
    }
    s.o.p();
}
```

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

```
for(i=1; i<=5; i++)
{
   for(i=1; j<=i; j++)
   {
      s.o.p(i);
   }
   s.o.p();
}
```

Array

class

→ class is a Rule ore structure using that structure we give real life existance/implementation.

→ class is a template ore blueprint from which objects are created.

→ Everything we wish to represent in a Java program must be encapsulated in class

→ class defines the state & behaviour of the basic programming components known as objects.

→ classes creates objects & object use methods to communicate b/t them.

eg.	class	Properties/state	Behaviour
	Human	Height weight colors	eat talk sleep

Anatomy/Syntax of class

```
<access specifier> class class-name
   (can be public/default)
{
   data-type variable name;
   return-type methodname(P1, P2)
}
```

Arrays in Java

→ An array is a collection of similar type of elements which has contiguous memory location.

→ Array in Java is index-based. The first element of the array is stored at the 0th index, 2nd element is stored in 1st index and so on.

```
  0   1   2   3   4   5
| 25 | 56 | 79 | 53 | 45 | 10 |   int [] A
```

Advantages

i) code optimization : It makes the code optimized, we can retrieve ore sort the data efficiently.

ii) Random Access : we can get any data located at an index position

Disadvantages

Size limit :- we can store only the fixed size of element in the array.

→ It does not grow its size at runtime.

Syntax to write array

1) datatype[] arrayname;
   eg   int[] a;

2) arrayname = new datatype[size];
   eg   regdno = new int[5];

3) datatype[] arrayname = new datatype[size];
   eg   int[] regd = new int[5];

Types of Array

Arrays are classified into two types.

1) Single Dimensional (1D) []
2) Multidimensional [][]

Single Dimensional :-

1) class A
   {
     public static void main(String args[])
     {
       int a[] = new int[5];
       a[0] = 10;
       a[1] = 20;
       a[2] = 30;
       a[3] = 40;
       a[4] = 50;

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

Direct Initialization

public class A
{
  public static void main(String args[])
  {
    int a[] = {33, 3, 4, 5, 4};

    for(int i = 0; i < a.length; i++)
    {
      S.o.p(a[i]);
    }
  }
}

For-each loop for Java Array

we can also print the Java array using for-each loop.

Syntax

for(data-type : array)
{
  body;
}

*4) Print array elements one by one

```
class Arrayexp
{
    public static void main(String args[])
    {
        int arr[] = {33, 11, 22};
        for(int i : arr)
            System.out.println(i);
    }
}
```

User input :→

```
import java.util.Scanner;
class Test_array
{
    public static void main(String args[])
    {
        int []ah = new int[5];
        S.o.p("Enter value to array");
        Scanner sc = new Scanner(System.in);
        for(int i = 0; i < 5; i++)
        {
            ah[i] = sc.nextInt();
        }
        System.out.println("values of array ");
        for(int i = 0; i < 5; i++)
            S.o.p(ah[i]);
    }
}
```

Demonstration of character Array

```
class char_array
{
    public static void main(String args[])
    {
        char []ah = {'m', 'e', 'g', 'h', 'a'};
        System.out.println("values of array ");
        for(int i = 0; i < 5; i++)
        {
            System.out.println(ah[i]);
        }
    }
}
```

finding Maximum Element in an Array

```
import java.util.Scanner;
class Max_array
{
    public static void main(String args[])
    {
        int []g = new int[5];
        System.out.println("Enter value to array");
        Scanner sc = new Scanner(System.in);
        for(int i = 0; i < 5; i++)
        {
            g[i] = sc.nextInt();
        }
        int max = g[0];
        for(int i = 0; i < 5; i++)
            if(max < g[i])
```

```
        { max = g[i];
        }
    }

    S.o.p("The max element of the
    array is" + max);
    }
}
```

Finding Sum of the element of an
Array

```
import java.util.Scanner;
class Sum_array
{
    public static void main(String args[])
    {
        int[] g = new int[5];
        System.out.println("Enter values");
        Scanner sc = new Scanner(System.in);
        int i;
        for(i = 0; i < 5; i++)
        {
            g[i] = sc.nextInt();
        }
        int sum = 0
        for(i = 0; i < 5; i++)
        {
            sum = sum + g[i];
        }
        S.o.p("The sum is" + sum);
```

```
    }
}
```

Multidimensional Array

In Java multi dimensional arrays are
actually array which is achieved by
adding more square bracket.

→ A 2D array can be like a table or
matrix of rows and columns.

Syntax

a) Declaration Syntax
   datatype [ ][ ] arrayname = new datatype[si][si]
   eg:-
   int[ ][ ] a = new int[2][2];

b) Initialization
   arrayname[row][column] = value;
   eg   arr[0][0] = 1;

c) Direct Method
   int[ ][ ] arrayname = {{ }, { }};
   int [ ][ ] a = {{2,3},{4,5}};
   import java.util.Scanner;
   class TWOD_array
   {
       public static void main(String args[])
       {
           int[ ][ ] a = new int[3][2];
```

```java
Scanner sc = new Scanner(System.in);
S.O.P("Enter the element of matrix");
for(int r=0; r<3; r++)
{
    for(int c=0; c<2; c++)
    {
        a[r][c] = sc.nextInt();
    }
}
S.O.P("The elements are");
for(int r=0; r<3; r++)
{
    for(int c=0; c<2; c++)
    {
        S.O.P(a[r][c] + " ");
    }
    S.O.P();
}
```

a[0][0]	[0][1]
2	1
[1][0]	[1][1]
4	3
[2][0]	[2][1]
5	4

```java
class Testarray
{
    public static void main(String arg[])
    {
        int arr[][] = {{1,2,3},{2,4,5},{3,4,6}};
        for(int r=0; r<3; r++)
        {
            for(int c=0; c<3; c++)
            {
                S.O.P(arr[r][c] + " ");
            }
            S.O.P();
        }
    }
}
```

Addition of 2 Matrices in Java

```java
class Test_array2
{
    public static void main(String arg[])
    {
        int a[][] = {{1,3,4},{3,4,5}};
        int b[][] = {{1,3,4},{3,4,5}};
        int c[][] = new int[2][3];
        for(int r=0; r<2; r++)
        {
            for(int c=0; c<3; c++)
            {
                c[r][c] = a[r][c] + b[r][c];
                S.O.P(c[r][c] + "n");
            }
            S.O.P();
        }
    }
}
```

$$\begin{bmatrix} 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}_{2\times3} + \begin{bmatrix} 1 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}_{2\times3}$$

$$= \begin{bmatrix} 2 & 6 & 8 \\ 6 & 8 & 10 \end{bmatrix}_{2\times3}$$

Multiplication of Matrices

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 1*1+1*2+1*3 & 1*1+1*2+1*3 & 1*1+1*2+1*3 \\ 2*1+2*2+2*3 & 2*1+2*2+2*3 & 2*1+2*2+2*3 \\ 3*1+3*2+3*3 & 3*1+3*2+3*3 & 3*1+3*2+3*3 \end{bmatrix}$$

$$= \begin{bmatrix} 1+2+3 & 6 & 6 \\ 2+4+6 & 12 & 12 \\ 18 & 18 & 18 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \\ 18 & 18 & 18 \end{bmatrix}$$

```java
public class matrixmul
{
    public static void main (String args[])
    {
        int [][]a= {{1,1,1},{2,2,2},{3,3,3}};
        int [][]b= {{1,1,1},{2,2,2},{3,3,3}};
        int c[][] = new int [3][3];

        for (i=0; i<3 ; i++)
        {
            for(j=0; j<3; j++)
            {
                c[i][j] = 0;
                for(int K=0; K<3; K++)
                {
                    c[i][j]+= a[i][K] * b[K][j]
                }
                S.o.p(c[i][j] + " ");
            }
            S.o.p(" ");
        }
    }
}
```

Object

→ Instance of a class is called object

→ An object in Java is essentially a block of memory that contains space to store variable (class does not reserved memory).

→ objects in Java are created using new operator
    ↳ used to assign memory

Syntax for object creation

① classname objectname ;
    objectname = new classname();

② classname object = new classname();
        name

eg :- class A
   {
     static int n, m;
     public static void main(String args[])
     {
      →A ob = new A();
      ob.n = 10;
      ob.m = 20;
      System.out.println(+n);
      System.out.println(m);

Methods in Java

→ A method is a block of code which only runs when it is called.

→ you can pass data, known as parameters/ arguments.

→ Methods are used to perform certain actions & they are also know as functions.

create a Method

→ A method must be declared within a class.

→ It is defined with the name of the method, followed by parenthesis ().

Syntax
                    ← may/may not
access   return   method (parameters...)
specifier type   name

eg public int add(int a, int b)

Addition of 2 no. using Method

Public class method_add
{
  public static void add()
  {
    int a = 20, b = 50;
    int c = a+b;

```
System.out.println " The sum is :" +c);
    }
    public static void main (String args[])
    {
        add();
    }
}
```

Types of Method

There are 2 types of method

a) Predefined Method
b) Userdefined Method

a) Predefined / Built-in Method :-

The method i.e already defined in Java class libraries.

→ We can directly use that method just by calling them in the program at any point.

eg → length(), equal(), compareTo(), Sqrt(), max() etc

↳ concat (): It concatenate 2 strings
    S1.concat (S2);

↳ charAt(): Returns a character by index position
    S1.charAt(1);

↳ toUpperCase() :- convert value to upper case
    S1.toUpperCase();

↳ tolowercase() :- convert value to lowercase
    S1.toLowerCase();

↳ trim(): Remove space from both side of a string. eg S1.trim();

↳ abs(): Return absolute value
    eg Math.abs(value);

↳ round(): It round the value to nearest integer.
    eg Math.round(a);
                    ↑value

↳ min() / max():

- Access Modifier in Java :-

→ Access modifier in java helps to restrict the scope of a class, constructor, variable, method or data member.

→ It specifies the accessibility

→ There are 4 types of the access modifier used in Java

1) private
2) default
3) public
4) Protected

1 - Private :→

The access level of a private modifier is only within the class. It can't be accessed from outside the class.

→ A subclass / childclass does not inherit the private members of its parent class.

B. java

```
class A
{
    private int d = 40;
    private void msg()
    {
        System.out.println("Hello");
    }
}

class B
{
    public static void main(String args[])
    {
        A ob = new A();
        System.out.println(ob.d); // compile Time Error
        ob.msg(); // compile time error
    }
}
```

note :→

If you make any class constructore private, you can not create the instance of that class from outside of the class

constructore :-

→ It is block of codes similar to the method. It is called when instance of a class is created.

B.java → It is a special type of method which is used to initialize the object

```
class A
{
    private A()
    {
        void msg()
        {
            System.out.println("Hello");
        }
    }
}

public class B
{
    public static void main(String args[])
    {
        A ob = new A(); // compile Time error
    }
}
```

Note

A class can not be private or protected except Nested class.

2) Default :-

If we don't use any modifier, it is treated as default. by default.

→ The default modifier is accessibly only within the program.

→ In the below example, we have created two packages pack & mypack. we are

accessing the class A from outside its package, since A class is not public. So it cannot be accessed from outside the package

A.java

```
Package pack;
class A
{
    void msg()
    {
        System.out.Println("Hello");
    }
}
```

B.java

```
package mypack;
import pack.*;
class B
{
    psvm(...)
    {
        A ob = new A();
        ob.msg();
```

3) Protected

→ The protected access modifier is accessible through inheritance only.

→ The protected modifier can be applied on the data member, method & constructor.

* It can't be applied on the class.

A.java

```
class A
{
    Protected void msg()
    {
        System.out.Println("Hello");
    }
}
```

B.java

```
class B extends A
{
    public static void main(String args[])
    {
        B ob = new B();
        ob.msg();
    }
}
```

4) public

The public access modifier is accessible everywhere.

→ It has the widest scope among all other modifiers.

A.java

```
class A
{
    public void msg()
    {
        System.out.Println("Hello");
    }
}
```

B.java

```
class B extends A
{
    public static void main(String args[])
    {
        B ob = new B();
        ob.msg(); }}
```

word

All Java Access Modibiere

Access modibire	within class	within Package	Outside pckg by subcds only (inheritance)	outside Packag
Private	yes	No	No	No
Detault	yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	YRS	Yes	Yes	Yes

String Builder So Buffer :→

String

→ In Java, String is a sequence of chriactors.

→ String is basically an object that represents sequence of char values.

→ The java.lang. String class is used to create String object.

19

```
String S1 = "Java";
String S2 ;
S2 = S1;
System.out. Println(S2);
```

```
class Test-string
{
  public static void main(String args[])
  {
    char [] ch = {'H','e','c','i','o'};
    for(int i=0; i<ch.length; i++)
    {
      System.out.println(ch[i]);
    }
  }
}
```

```
class Test-string2
{
  public static void main(String args[])
  {
    String S1 = { "Hello";
    System.out.println(S1);
  }
}
```

How to create String object ?

There are 2 ways to create String object :

1. By String Literal
2. By new Keyword

1. **String Literal :**

   String S = "welcome";

→ String objects are stored in a Special memory area known as String constant pool.

2. **New Keyword :**

   String S = new String("Hello");

String_ex.java
public Class String_ex
{
    public static void main (String angs[])
    {
        String S1 = "Hello";
        Char ch[] ={'s','t','u','d','e','n','t','s'};
        String S2 = new String (ch);
                        ⇒ converting chan
                        → to string
        String S3= new String ("brood");
        System.out.println (S1);
        System.out.println (S2);
        System.out.println (S3);
    }
}

Java String Buffer class : ⇒

Java StringBuffer class is used to created mutable (modifiable) String.

→ The StringBuffer class in Java is same as string class except it is mutable i.e it can be changed.

Constructors of String Buffer class

1. StringBuffer(): Creates an empty string buffer with the initial capacity 16

2. StringBuffer(String strs): Create a string buffer with the specified String

3. StringBuffer(int capacity): create an empty string buffer with specified capacity as length.

◉ Mutable String : ⇒

→ A string that can be modified or changed is known as mutable String.

1) append() method

The append() method concatenates the given argument with the String.

class A
{
    public static void main(String angs[])
    {
        StringBuffer Sb = new StringBuffer ("Hello");
        Sb.append ("Brood Monning");
        System.out.println (Sb);
    }
}

2) StringBuffer insert() method :

The insert() method inserts the given String with this String at the given position.

```java
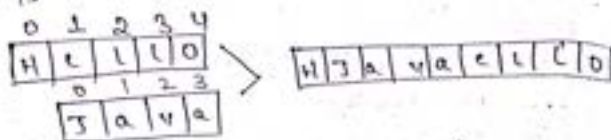class Str_insert
{
    public static void main(String args[])
    {
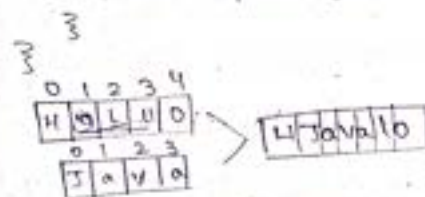        StringBuffer sb = new StringBuffer("Hello");

        sb.insert(1, "Java");
        System.out.println(sb);
    }
}
```

```
 0 1 2 3 4
┌─┬─┬─┬─┬─┐              ┌─┬─┬─┬─┬─┬─┬─┬─┬─┐
│H│e│l│l│O│      →       │H│J│a│v│a│e│l│l│O│
└─┴─┴─┴─┴─┘              └─┴─┴─┴─┴─┴─┴─┴─┴─┘
 0 1 2 3
┌─┬─┬─┬─┐
│J│a│v│a│
└─┴─┴─┴─┘
```

3) StringBuffer replace() method

Str_replace.java

The replace() method replace the given string from the specified beginIndex & endIndex

```java
class Str_replace
{
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.replace(1, 3, "Java");
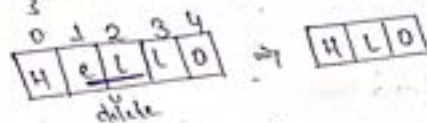        System.out.println(sb);
    }
}
```

```
 0 1 2 3 4
┌─┬─┬─┬─┬─┐              ┌─┬─┬─┬─┬─┬─┐
│H│e│l│l│O│      →       │L│Java│lo│
└─┴─┴─┴─┴─┘              └─┴─┴─┴─┴─┘
 0 1 2 3
┌─┬─┬─┬─┐
│J│a│v│a│
└─┴─┴─┴─┘
```

4) delete() method:

The delete() method of StringBuffer class delete the string from the specified beginIndex to endIndex

Str_delete.java

```java
class Str_delete
{
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.delete(1, 3);
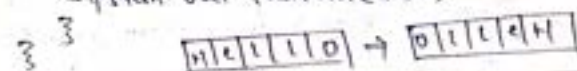        System.out.println(sb);
    }
}
```

```
 0 1 2 3 4
┌─┬─┬─┬─┬─┐              ┌─┬─┬─┐
│H│e│l│l│O│      →       │H│l│O│
└─┴─┴─┴─┴─┘              └─┴─┴─┘
     delete
```

5) reverse() method

The reverse() method of StringBuffer class reverse the current string.

Str_reverse.java

```java
class Str_reverse
{
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.reverse();
        System.out.println(sb);
    }
}
```

```
┌─┬─┬─┬─┬─┐       ┌─┬─┬─┬─┬─┐
│H│e│l│l│O│   →   │O│l│l│e│H│
└─┴─┴─┴─┴─┘       └─┴─┴─┴─┴─┘
```

Difference bet^n String Buffer & Builder

String Buffer	String Builder
→ StringBuffer is Synchronized i.e thread Safe. It means two threads can't call the methods of StringBuffer Simultaneously.	→ StringBuilder is non-Synchronized
→ Less efficient	→ More efficient

Parameter passing:
There are different ways in which parameter data can be passed into & out of methods.

Types of Parameter

a) Formal Parameter
- A variable & its type as they appear in the prototype of the fun^n or method.

Syntax
fun^n name (data-type variable-name)

b) -Actual Parameter:
The variable or expression corresponding to a formal parameter that appears in the fun^n in the calling environment.

Syntax
Fun^n name (variable(s));

are values that are passed to a fun^n when it invoked

eg: void add(int x, int y) → Formal Parameter
```
{
    int Z = x + y;
    System.out.println ("Z);
}
add (2, 3);
add (5, 4);   → Actual Parameter
```

Important Method fore Parameter passing
1. Pass By Value / call by value :-
→ call by value means calling a method with a parameter as value.
→ The argument value is passed to the parameter

eg: Public static void main (String args[])
```
{
    Sum (1, 2);
}
Public static int Sum(int x, int y)
{
    return x + y;
}
```

Constructor :-

→ A constructor is a special member function / method / procedure / operation / behaviour / action of a class is used to create object of that class.

→ It has same name as the class itself.

→ It has no return type & is invoked or called using new operation.

Syntax :-

```
modifier classname (datatype variable...)
{
    Statement 1;
    :
    Statement n;
}
```

A.java

19
```
public class A
{
    public static void main(String args[])
    {
        A ob = new A();
    }
    A()
    {
        System.out.println("constructor is
                            called");
    }
}
```

```
public class A1
{
    A1()
    {
        System.out.println("Hello");
    }
    public void show
    {
        System.out.println("Good morning");
    }
    public static void main(String args[])
    {
        A1 ob = new A1();
        ob.show();
    }
}
```

Types of Java constructors :-

There are two types of constructors in Java.

1. Default constructor
2. Parameterized constructor

1. Default constructor :-

→ A constructor is called default constructor when it does not have any parameter.

Syntax

```
classname()
{
    ≡
}
```

add.java

```java
1) class add
   {
     add()
     {
       int c=10+20;
       System.out.println(c);
     }
     public static void main(String args[])
     {
       add ob=new add()
     }
   }
```

purpose

It is used to provide the default value to the object like 0, null etc

```java
2) class Student
   {
     int Regd;
     String name;
     void display()
     {
       S.o.p( Regd+" "+name);
     }
     public static void main(String args[])
     {
       Student S1 = new Student();
       Student S2 = new Student();
       S1.display();
       S2.display();
     }
   }
```

Note: If there is no constructor in a class, compiler automatically creates a default constructor.

2. Parameterized constructor:-
—A constructor which has a specific number of parameter is called a parameterized constructor.

Add.java

```java
1) public class Add
   {
     Add(int m,int n)
     {
       System.out.println(m+n);
     }
     public static void main(String args[])
     {
       Add ob=new Add(10,20);
     }
   }
```

Add1.java

```java
public class Add1
{
  Add1(int m,int n)
  {
    System.out.println(m+n);
  }
  void add(int a,int b)
  {
    System.out.println(a+b);
  }
}
```

```
{
    Addl ob=new Addl(10,20);
    ob.add(20,30);
}
}
```

constructor overloading

constructore overloading in java is a technique of having more than one constructore with different parameter lists.

eg
```
class Student
{
    int id;
    String name;
    int age;
    Student (int i, String c)
    { id = i; name = c;
    System.out.println( id+ "  " + name);
    }
    Student (int i, String s, int ag).
    {
        id = i;
        s = name;
        ag = age;
    System.out.println( id+" "+ name + " " +
                                        age);
    }
}
```

```
Public static void main(String args[])
{
    Student s1= new Student(1, "Megha");
    Student s2= new Student(2, "Monali", 20)
}
}
```

comparing & Identifying object

→ In Java, the == operatore compares that two refereences are identical or not, where whereas the equals() method compares two objects.

→ objects are equal when they have the same state. objects are identical when they share the class identity.

eg obj1 == obj2 (test identity)
    obj.equals(obj2) (compares equality)

Inheritance in Java :-

→ Inheritance in Java is a method mechanis in which one object acquires all the properties & behaviour of parent class.

→ The idea behind inheritance in java is the you can create new classes that are built upon existing classes.

→ when we inherit from an existing class. we can reuse method & fields of the parent class. we can add new method

field in oure current class.

→ Inheritance represents the Is-A relationship which is also known as Parent-child relationship

Important terminology

i) **Super class** :-
The class whose features are inherited is known as Superclass (ore base class/parent class).

ii) **Sub class** :
The class that inherit the other class is known as subclass (ore derived class/extended class/child class)

Syntay
The keyword used fore inheritance is "extends"

→ indicates that you are making a new class that derives from an existing class.

```
class childclass extends Parentclass
{
   codes
}
```

class parentclass
{
   -
}
class childclass extends Parentclass
{
   -
}

B.java

eg class A
{
   int a=10, b=20;
}
class B extends A
{
   public static void main(String args[])
   {
      int c=a+b;
      System.out.println("The sum is;"+c);
   }
}

Types of Inheritance:

→ on the basis of class, there can be 5 types of inheritance in Java.
→* Multiple & Hybrid inheritance supported through interface only)

1. Single Inheritance.
2. Multilevel Inheritance
3. Hierarchical Inheritance

1) Single Inheritance

2. Multilevel

3. Hierarchical

4. Multiple

5. Hybrid

1) **Single Inheritance:**

when a (only 1 class) class inherite another class, is known as single Inheritance

Syntax

```
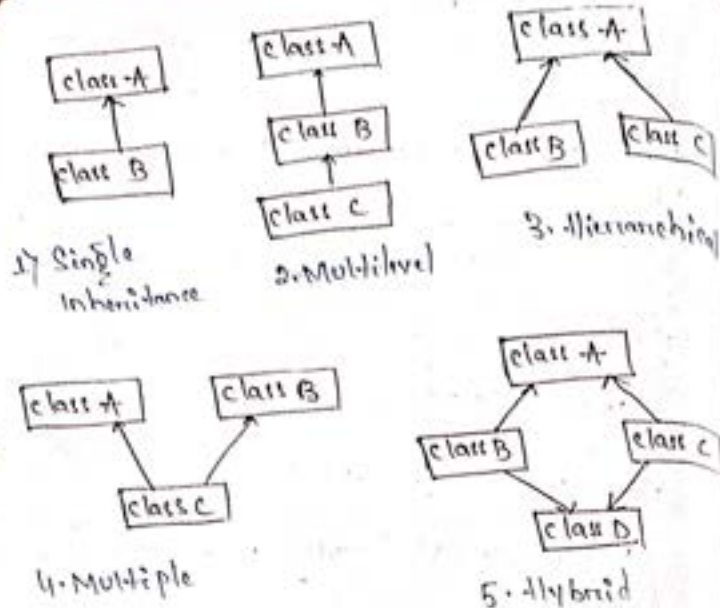public class A
{
   ≡
}
public class B extends A
{
   ≡
}
```

```
public class A
{
   public void msg()
   {
      System.out.println("Hello");
   }
}
public class B extends A
{
   public static void main(String args[])
   public void display()
   {
      System.out.println("Good Morning");
      System.out.println("How are you?");
   }
   B ob = new B();
   ob.msg();
   ob.display();
}
```

2. **Multilevel inheritance**

When there is a chain, it is called multilevel inheritance

Syntax
```
public class A
{ ≡
}
public class B extends A
{ ≡
}
public class C extends A B
{ - }
```

```
class X
{
    public void ad mY()
    {
        System.out.println(" X method");
    }
class Y extends X
{
    public void my()
    {
        S.o.p("Y method");
    }
}
class Z extends Y
{
    public void mZ()
    {
        System.out.println("Z method");
    }
    public static void main(String args[])
    {
        Z ob = new Z();
        ob.mY();
        ob.my();
        ob.mZ();
    }
}
```

5. Hierarchical class :

If one class is inherited by many subclasses i.e. known as hierarchical class.

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

```
class X
{
    public void nX()
    {
        S.o.p(" X method");
    }
}
public class Y extends X
{
    public void MY()
    {
        S.o.p("Y method");
    }
}
public class Z extends X
{
    public void MZ()
    {
        S.o.p("Z method");
    }
    public static void main(String args[])
    {
        Y ob = new Y();
        X ob1 = new X();
        ob.mX();
        ob.My();
        ob1.MX();
        ob1.MY();
    }
}
```

* Java does not support multiple Inheritance. This means that class cannot extend more than one class.

The Super keyword

→ The Super keyword is used to differentiate the members of Superclass/Base class from the members of Subclass/childclass, if they have same name.

Differentiating the members

If a class is inheriting the properties of another class. And if the members of the Superclass have the name & some as the Sub class, to differentiate these variables we use Super keyword

- Super.variable();
  Super.method();

child.java

```
class Parent
{
    int n = 10;
    public void display()
    {
        System.out.println("This is Super class");
    }
}

class child extends Parent
{
    int n = 20;
    public void display().
    {
        System.out.println("This is Sub class");
    }
    public void my_method()
    public static void main(string args())
```

```
        child ob = new child();
        ob.display();
        8 super.display();
        S.o.p( ob.num);
        S.o.p( super.num);
    }
    public static void main(String args())
    {
        child obj = new child();
        obj.my_method();
    }
}
```

Note

1) Super can be used in method which is non-static.

2) In the above example super is used in a non-static method named as my_method.

Polymorphism

It is an ability of an object to take on many form..

- int x = 10, y = 64;
  System.out.println(" +x"); // a 379 as string
  System.out.println(" x" = + x);
  S.o.p(" x" = + x + " y" = + y);

Two Types of polymorphism
a) compile Time polymorphism(static or Early Binding)
b) Run Time(Dynamic or Late Binding) or Dynamic Dispatching

a) Compile Time :
The object bind/ call/invoke/ associated with the appropriate method is already determined ore decided priore to the compilation.

b) Run Time
The object bind/ call /invoke/ associate with the appropriate method is determined ore decided at run time only.

```
class calculation
{
    void add (int a, int b)
    {
        System. out. println( a+b);
    }
    void add (int a, int b, int c)
    {
        System.out. println( a+b+c);
    }
    public static void main (String args[])
    {
```

```
        Calculation ob = new Calculation();
        ob. add(10,20,30);
        ob. add(20,30);
    }
}
```

→ There are 3 ways to achieve polymor-phism
1. overloading
2. Overriding
3. Interfacing

1. Method overloading
Multiple methods by same name but different parameters, is known as Method overloading.

→ compiler knows which method/ function to call based on the parameter it is passed.

why?
Suppose we have to perform addition of the given number but there can be any no. of arguments, if we write the method such as add2 (int,int) fore two parameters and add3 (int, int,int) fore 3 arguments then it may difficult

for programmers to understand behaviour so we perform method overloading.

Two ways to overload method
1. By changing number of arguments
2. By changing the datatype

note
Method overloading is not possible by changing the return type of the method.

By changing no. of arguments

```
class calculation
{
  void add (int a, int b)
  {
    System.out.println (a+b);
  }
  void add (int a, int b, int c)
  {
    System.out.println( a+b+c);
  }
  public static void main (String args[])
  {
    calculation ob=new calculation();
    ob.add(10,20);
    ob.add(10,20,30);
  }
}
```

By changing data-type

```
class A
{
  void add (int a, int b)
  {
    System.out.println(a+b);
                double a, double b
  void add (int a, int b, int c)
  {
    System.out.print (a+b);
  }
  public static void main (String args[])
  {
    A ob = new A();
    ob.add(10,20);
    ob.add(10.5, 20.5);
  }
}
```

2. Method overriding → Same name, Same Parameters method

If Subclass has the same name as the parent class is known method overriding

Rules
1. Method must have same name as in the parent class.
2. Method must have Same parameters as in the parent class.

29.
```
class V
{
  void run ()
  {
    S.O.P (" Inside Parent class ");
  }
}

class B extends V
{
  void run ()
  {
    S.O.P (" Inside child class ");
  }
  public static void main (String args[])
  {
    B obj = new B();
    obj. run();    //    B obj2 = new B();
  }              //    obj2 . run();
}
```

Pass by value & Pass By Reference

→ Pass by value is also called call by value

→ Pass by Reference is also called call by Reference.

19.
```
public class B
{
  public void twofold (int n)
  {
    n = 2 * n;
    S.O.P (" value inside twofold method = "
         + n);     // 20
  }
  public void demo()
  {
    int x = 10;
    S.O.P (" value before twofold method call = " + x);   //11 10
    twofold (x);
    S.O.P (" value after twofold method = " + x);
  }              // 10
}
```

output

value before twofold method call = 10
value inside twofold method = 20
value after twofold method = 10

Call By Reference

-Arguments are passed by reference rather than passing by value.

```java
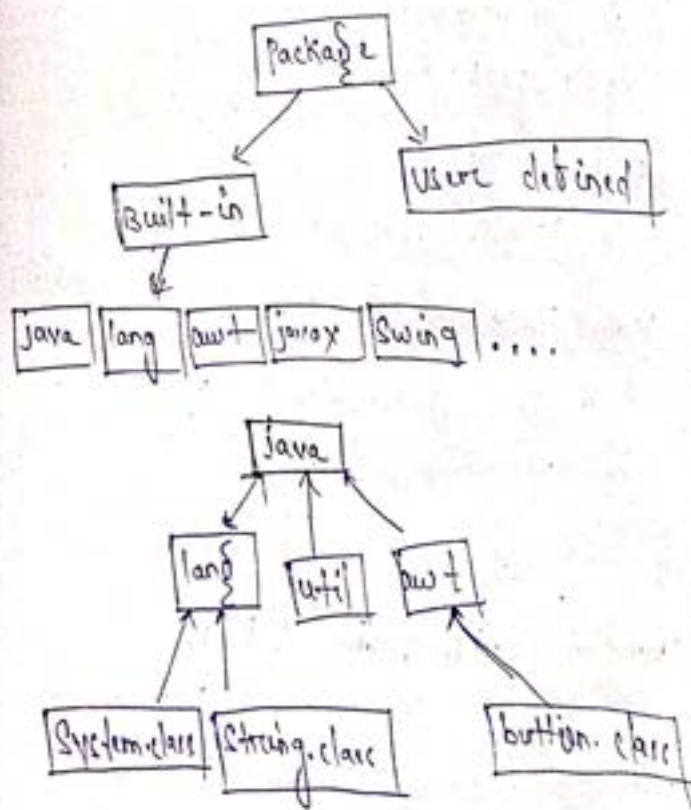class callByRef
{
    static int x=1, y=2, s;
    public void Sum()
    {
        S.o.p("original value" +x + " " +y);
    }
    s= x+y;
    S.o.p("original sum = " +s);
    callByRef ob= new callByRef();
    Test(ob);
    S.o.p("value after call" +x +" "+y);
}
public void Test(callByRef ob)
{
    callByRef.x = 100;
    callByRef.y = 200;
    int p= x*y;
    S.o.p("x*y" = " + p);
}
```

JAVA PACKAGE

→ Packages are used in java in order to prevent naming conflict, to control access, to make searching / locating & usage of classes, interfaces, enum etc easier.

→ A package can be defined as a grouping of related types (classes, interfaces etc), providing access protection & namespace management.

→ Package helps us to categorize our code in different folders.

```
          ┌──────────┐
          │ Package  │
          └──────────┘
           ↙        ↘
    ┌─────────┐   ┌──────────────┐
    │ Built-in│   │ User defined │
    └─────────┘   └──────────────┘
         ↓
┌────┐┌────┐┌────┐┌─────┐┌──────┐
│java││lang││awt ││javax││Swing │....
└────┘└────┘└────┘└─────┘└──────┘


          ┌──────┐
          │ java │
          └──────┘
         ↙    ↑    ↘
    ┌──────┐┌──────┐┌──────┐
    │ lang ││ util ││ awt  │
    └──────┘└──────┘└──────┘
      ↑  ↑              ↑
┌────────────┐┌────────────┐  ┌────────────┐
│System.class││String.class│  │button.class│
└────────────┘└────────────┘  └────────────┘
```

→ The "package" keyword is used to create a package in java.

How to compile Java Package

javac -d directory javafilename
     ‿‿
     switch to specifies the destination
where to put the generated class.

‣ javac -d. Simple.java

‣ # (.) package within the same directory

javac -d F:\Java\ pck Simple.java

How to Run Java Package

java packagename.classname
java mypack.Simple

                    Simple.java
package mypack;
public class Simple
{
  public static void main(String args[])
  {
    System.out.Println("welcome");
  }
}

changing drive from c to D
:\)D:

How to Access Package from Another Package?

• There are 3 ways to access package from outside the package.

1. import package.*;
2. import package.classname;
3. Fully Qualified name

1] Using Package Name:

If we use package.*, then all the classes & interfaces of this package will be accessible but not Sub package.

→ The import keyword is used to make classes & interfaces of another package accessible to the current package.

                    A.java
package testpack;
public class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
                    B.java
package mypack;
import testpack.*;
public class B
{

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

compile

F:\Java\PcK> javac -d F:\Java\PcK A.java
F:\Java\PcK> javac -d F:\Java\PcK B.java
Run F:\Java\PcK> java mypack.B

2) using packagename.classname

IF we import packagename.classname only declared class of this package will be accessible.

e.g:-   A.java
```
package testpack;
public class A
{
    public void msg()
    {
        System.out.println("Hello");
    }
}
```
        B.java
```
package mypack;
import testpack.A;
public class B
{
```

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

3) using Fully Qualified Name :
IF we use fully Qualified name then only declared class of this package will be accessible. Now there is no need to import.

        A.java
```
package testpack;
public class A
{
    public void msg()
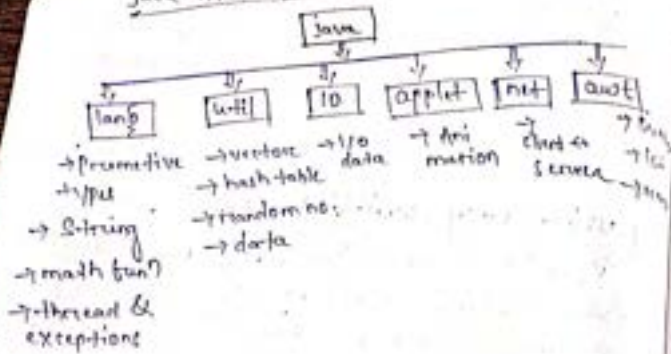    {
        System.out.println("Hello");
    }
}
```
        B.java
```
package mypack;
public class B
{
    public static void main(String args[])
    {
        testpack.A ob = new A();
        ob.msg();
    }
}
```

Java → API Package

```
            ┌──────┐
            │ Java │
            └──────┘
   ↓      ↓      ↓      ↓       ↓      ↓
┌─────┐ ┌────┐ ┌──┐ ┌───────┐ ┌────┐ ┌─────┐
│ lang│ │util│ │IO│ │applet │ │ net│ │ awt │
└─────┘ └────┘ └──┘ └───────┘ └────┘ └─────┘
```

→ Primitive   → vector   → I/O   → Ani   client   →
   type   → hash-table   data   mation   server

→ String   → random no:

→ math fun?   → date

→ thread &
exceptions

JVM (Java virtual Machine) :

→ JVM is a specification that provides run-
time environment in which java byte-code
can be executed.

What it does ?

The JVM performs following operation

→ Loads code
→ Verifies code
→ Executes code
→ provides runtime environment

JVM — Architecture

→ It contains classloader, memory area,
execution engine etc

Java Runtime │classloader│ → Memory Area
System   Allocated by
                 ⇓   JVM

```
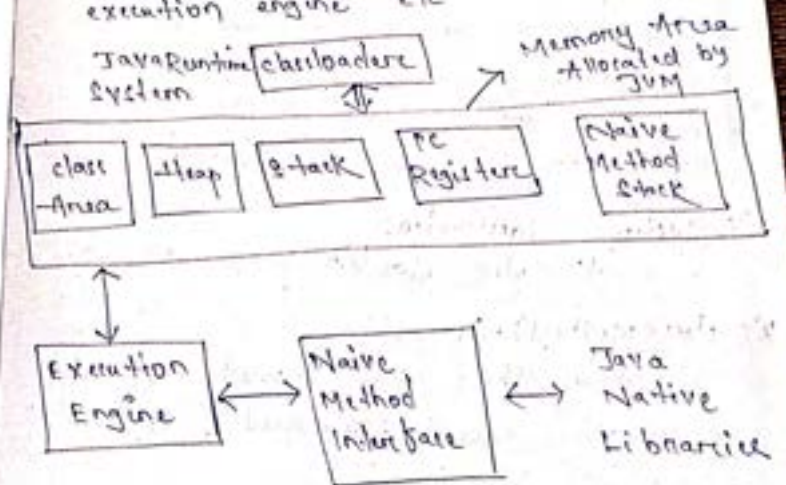┌───────┐ ┌──────┐ ┌──────┐ ┌──────────┐ ┌────────┐
│ class │ │ Heap │ │ Stack│ │ PC       │ │ Naive  │
│ Area  │ │      │ │      │ │ Register │ │ Method │
│       │ │      │ │      │ │          │ │ Stack  │
└───────┘ └──────┘ └──────┘ └──────────┘ └────────┘
```

```
┌──────────┐       ┌─────────┐        ┌──────────┐
│Execution │ ←───→ │ Naive   │ ←───→  │ Java     │
│ Engine   │       │ Method  │        │ Native   │
│          │       │Interface│        │ Libraries│
└──────────┘       └─────────┘        └──────────┘
```

1) class Loaders:
- It is a subsystem of JVM which is used to load class files.
→ Whenever we run the java program, it is loaded first by the classloader.
→ There are 3 built-in classloaders in Java.

a) Bootstrap classLoader:
Super class of extension classloader.
→ It loads rt.jar file which contains all class files of java standard like like java.lang, java.net, java.io, java.sql etc

b) Extension classLoader:
This is the child classloader of Bootstrap & parent classloader of System classloader.
→ It loads the jar files located inside $Java-HOME/jre/lib/ext directory

c) System classLoader:
It loads classfiles from classpath.

2) class (Method) Area:
Stores Per-class structures such as runtime const. pool, field & method data, code for method.

3) Heap:
Run time data area in which objects are allocated.

4) Stack:
It holds local variables & partial results, and plays a part in method invocation & return.

5) PC Register:
contains address of the JVM instruction currently being executed.

6) Native Method Stack:
It contains all the native methods used in the appln.

7) Execution Engine:
It contains:
a) A virtual processor
b) Interpreter
c) JIT (Just-In-Time): JIT compiles parts of the byte code that have similar funn at the same time

8) Java Native Interface:
It is a framework which provides an interface to communicate with another appln written in another language like c, c++ etc.

Exception Handling in JAVA

Error:

Errors are the wrong that make a program go wrong.

Compile - Time Errors:

Syntax errors at compile time because of

→ Missing Semicolons

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

→ Missing (or mismatch of) brackets in class & method body.
→ misspelling of identifiers & keywords
→ missing double quotes in strings.
→ use of undeclared variables
→ Incompatible types in assignment
→ Bad reference to object ....etc

Reading Data from keyboard

There are many ways to read data from keyboard. For example

a) InputStreamReader
b) Scanner
c) console/command. Line Arguments
d) DataInputStream.

a) InputStreamReader class :

It can be used to read data from keyboard.

→ It performs 2 tasks.
   i) connect to input stream of keyboard
   ii) converts the byte oriented stream into character - oriented stream

BufferReader class :

It can also be used to read data line by line by readLine() method.

```
import java.io.*;
class A
{
    public static void main (String args[])
    {
        InputStreamReader r=new InputStreamReader(System.in);
        BufferReader b= new BufferReader (r);
        System.out.println (" Enter your Name");
        String name= b.readLine ();
        S.O.P(name);
    }
}
```

b) **Java Scanner class:**

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them....

→ The Java Scanner class breaks the input into token using a delimiter that is whitespace by default.

Commonly used Method of Scanner class

i) public String next():—
It returns the next token from the Scanner.

ii) public String nextLine():—
It moves the Scanner position to the next line & returns the value as a String.

iii) public byte nextByte():—
It scans the next code as Byte

iv) public int nextInt():—
It scans the next token as Integer.

v) public long nextLong():—
It scans the next token as Long.

vi) public float nextFloat():—
It scans the next token as Float.

vii) public double nextDouble():—
It scans the next token as Double.

```
import java.util.Scanner;
class S
{
  psvm(....)
  {
    Scanner sc = new Scanner(System.in);
    S.O.P("Enter your Name");
    String name = sc.next();
    S.O.P("Name is:" + name);
  }
}
```

Java Scanner example with Delimiter(white space):

```
import java.util.*;
class S1
{
  psvm(..)
  {
    String input = 20 tea 20 coffee;
    Scanner sc = new Scanner(input).
    useDelimiter("\\s");
    S.O.P(sc.nextInt());
    S.O.P(sc.next());
    S.O.P(sc.nextInt());
    S.O.P(sc.next());
    sc.close();
  }
}
```

→Addition of 2 no:

```java
import Java.util.Scanner;
class Input
{
    p.s.v.m(...)
    {
        Scanner Sc = new Scanner(System...);
        S.o.p("Enter 1st No.");
        int a = Sc.nextInt();
        S.o.p("Enter 2nd No.");
        int b = Sc.nextInt();
        int c = a+b;
        S.o.p(c);
    }
}
```

c) Command Line Argument as Input

Java main method consist array of Strings as main(String args[]). when we run the program, the array will be filled with the value of any arguments through command line.

→ Each element in an array can be referenced or accessed by an arraynam & index (a[0]).

q c:> javac A.java

c:> java A 1 2 3

Here command line contains 3 arguments

```
a[0]    is    1
a[1]    is    2
a[2]    is    3
```

q) class A
{
    public static void main(String args[])
    {
        System.out.println("Hello");
        System.out.println(a[0]);
    }
}

compile :— c:...> javac A.java
Run :— c:...> java A Monali

o/p   Hello Monali

```java
class A1
{
    public static void main(String args[])
    {
        int i=0;
        for(i=0; i<a.length; i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Java IO :-

Java IO(input & output) is used to process the input & produce the output.

→ The java.io package contains all the classes required for input & output operations.

Stream

A Stream is a sequence of data.

→ In java, a stream is composed of bytes. It's called a stream because it is like stream of water that continues to flow.

→ In java 3 streams are created for us automatically.

1) **System.out** : Standard output Stream
2) **System.in** : Standard input Stream
3) **System.err** : Standard error Stream

eg System.out.println("Simple Message!");
System.err.println("error Message!");
int i = System.in.read();
System.out.println((char)i);

Output Stream :

Java appln uses an output Stream to write data to a destination.

→ It may be a file, an array, periphieral device.

Input Stream :

Java appln uses an input Stream to read data from a source; it may be a file.

outputStream class :

It is the superclass of all classes representing an output Stream of byte.

→ An output Stream accepts output bytes & sends them to some sink.

Useful Methods of outputStream

Method	Description
1) public void write(int) throws IOException	Is used to write a byte to the current output Stream
2) public void write(byte[]) throws IOException	is used to write an array of byte to the current output Stream
3) public void flush() throws IOException	Flushes the current output Stream
4) public void close() throws IOException	is used to close the current output Stream

Inputstream class :

→ It is the superclass of all classes representing an input stream of bytes

Useful Methods of Inputstream

method	Description
1) public abstract int read() throws IOException	→ Reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available() throws IOException	→ returns an estimate of the no. of bytes that can be read from the current input stream
5) public void close() throws IOException	→ is used to close the current input stream

Java FileOutputStream class :-

Java FileOutputStream is an output stream used fore writing data to a file (write byte)

```
import java.io.FileOutputStream ;
public class FileoutputExample
{
   public static void main (String args[])
   {
      try
      {
```

```
FileOutputStream fout = new FileOutput
                    Stream
            ("D:\\t1.txt");
                      ↳ location of the
fout.write(65);                    file
fout.close();
System.out.println ("success !");
}
catch (Exception e)
{
   System.out.println(e);
}
}
}
```

t1.txt

-A

write String

```
import java.io.FileOutputStream ;
public class FileExample
{
   public static void main (String args[])
   {
      try
      {
         FileOutputStream fout = new FileOutputStream
                    ("D:\\t1.txt");
         String S = "welcome to my file";
         byte[] b = S.getBytes();
         fout.write(b);
         fout.close();
         System.out.println ("success"); {
```

```
        catche Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Java FileInputStream class

Java FileInput Stream class obtains input bytes from a file.

Java FileInputStream class Methods

Method	Description
int available()	It is used to return the estimated no. of bytes that can be read from the input Stream
int read()	It is used to read th. byte of data from the input Stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input Stream.
Void close()	It is used close the Stream

19 Read Single character

```
import java.io.FileInputStream;
public class DataStreamEX
{
    public static void main (String args[])
    {
        try
        {
            FileInputStream fin = new FileInputStre
                            ("D:\\-ll-.txt");
            int i = fin.read();
            System.out.println((char)i);
            fin.close();
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

19 Read all chnacters
```
import java.io.FileInputStream;
public class A
{
    public static void main (String args[])
    {
        try
        {
                                    new
            FileInputStream fin = FileInputStream("D:\\l-.
                                    txt");
            int i=0;
```

```
catch ( Exception e)
{
    System.out.println(e);
}
}
}
```

Java FileInputStream class

Java FileInput Stream class obtains input bytes from a file.

Java FileInputStream class Methods

Method	Description
int available()	It is used to return the estimated no. of bytes that can be read from the input stream
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input stream.
void close()	It is used close the stream

Read Single character

```
import java.io.FileInputStream;
public class DataStreamEX
{
    public static void main (String args[]
    {
        try
        {
            FileInputStream fin =new FileInputS
                ("D:\\tl.txt");
            int i = fin.read();
            System.out.println((char)i);
            fin.close();
        }
        catch ( Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Read all characters

```
import java.io.FileInputStream;
public class A
{
    public static void main (String args[])
    {
        try
        {
            FileInputStream fin = new FileInputStream("D:\\tl
                .txt");
            int i =0;
```

```
while((i = fin.read())! = -1)
{
    System.out.println((char)i);
}
fin.close();
}
catch(Exception e)
{
    System.out.println(e);
}
}
}
```

File Handling in Java

In Java, File Handling is done with the help of the File class of the java.io package.

File Handling operation

1. Creating a new file
2. Writing in a file
3. Reading an existing file
4. Deleting a file

→ To perform any of the above operations on a file in java, we need to create an object of the File classes

eg: import java.io.File;
File obj = new File("Filename.txt");
                         ↳ specify the name of the file

1. Creating a New file:
import java.io.File;
import java.io.FileWriter;
public class CNF
{
    public static void main (String args[])
    {
        File ob = new File("abc.txt");
        try
        {
            ob.createNewFile();
            S.O.P("Success full");
        }
```

```
catch ( IOException e)
{                      → if the file can't
  S.O.P (e);     created due some time
}
}
}
```

→ To create a file in java, you can use the createNewFile () method.

→ This method returns a boolean value i.e. T/F. (The method is enclosed in a try ... catch)

2. Write to a File.

→ We use fileWriter (It is used to write character - oriented data to a file) class & Reading / writing text file) together with its write() method & write some text to the file we created.

→ You should close it with the close() method.

```
import java.io.FileWriter ;
import java.io. IOException ;
Public class WriteToFile
{
  Public static void main (String args[])
  {
    try
    {
      FileWrite FO = new FileWrite ("al.txt");
```

```
FO.Write ("Tricky but fun enough");
FO.close();
S.O.P ("Successfull ");
catch( IOException e)
{
  S.O.P (" Err200 ");
  e. printStackTrace ();
}
}
}
```

5. Read a file :-

We use the Scanner class to read the contents of the file

```
import java.io.File;
import java.io.FileNotFoundException ;
import java.util.Scanner;
Public class A
{
  Psvm (String args[])
  {
    try
    {
      File obj = new File ("al.txt");
      Scanner myReader = new Scanner (obj)
      while ( myReader.has nextLine())
      {
        String data = myReader.nextLine();
        S.O.P (data);
      }
```

```java
        myReader.close();
    } catch(FileNotFoundException e)
    {
        S.o.p("Errnore");
    } e.printStackTrace();
    }
}
```

4. Deleting File :-

To delete a file, use the delete() method

```java
import java.io.File;
public class DeleteFile
{
    public static void main(String args[])
    {
        File obj = new File("a1.txt");
        if(obj.delete())
        {
            S.o.p("Deletedfile : "+ obj.getName());
        }
        else
        {
            S.o.p("Failed to delete the file");
        }
    }
}
```