Bhavitha. T

High Level.
object oriented

Progamming Language
- low [Mov R, R2]
- middle [C]
- high [Java, C#, .Net]

⇓

JAVA

Just Another Virtual Accelerator

KISS → Keep It Simple and Straight

Gift from trainer mam

Java - Just Another Virtual Accelerator

Codification Convert to

OCJP

Just Another Virtual Accelerator

bin
Environment variables

iamnandeshks@gmail.com
nandesh.ks -> Insta

Rules & Syntax which as Set of Instructions to write a Programs to build a Software is known as Programming Language.

Java (Sun Microsystems)

Features

=> Platform Independent byte code which can run on multiple systems
=> Open Source / Licence free / Free of Cost
=> Highly Secured
=> Backward Compatibility (New features can also used in old features)

Platform Independent

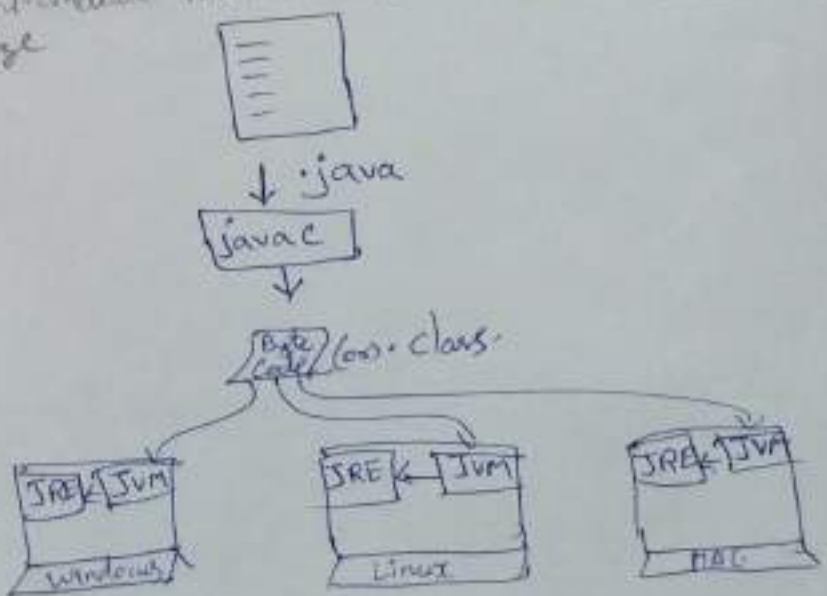
JDK ← Java Development Kit

- JavaC → Java Compiler
  → Converts Java file to Byte Code / .Class file
- JRE → Java Runtime Environment
  ~~→ As — an — Interp~~
  → It is a Platform for Execution
- JVM → Java Virtual Machine
  → It is an Interpreter, which Convert Byte Code to Machine Code (0, 1) + Execution

JDK is the Dependent One

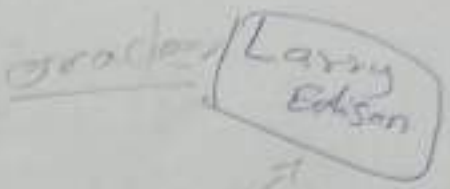
Compiler → Runs whole file and Converts to Byte code / .class

Generate Intermediate machine code
↓
large in size



Why Pointers or pointers Arithematic is Eliminated from Java?

i) Pointers leads Confusion for the Programmer
ii) Pointers will not Provides Data Seamity
iii) Pointer can be used to write Virus Programs



History

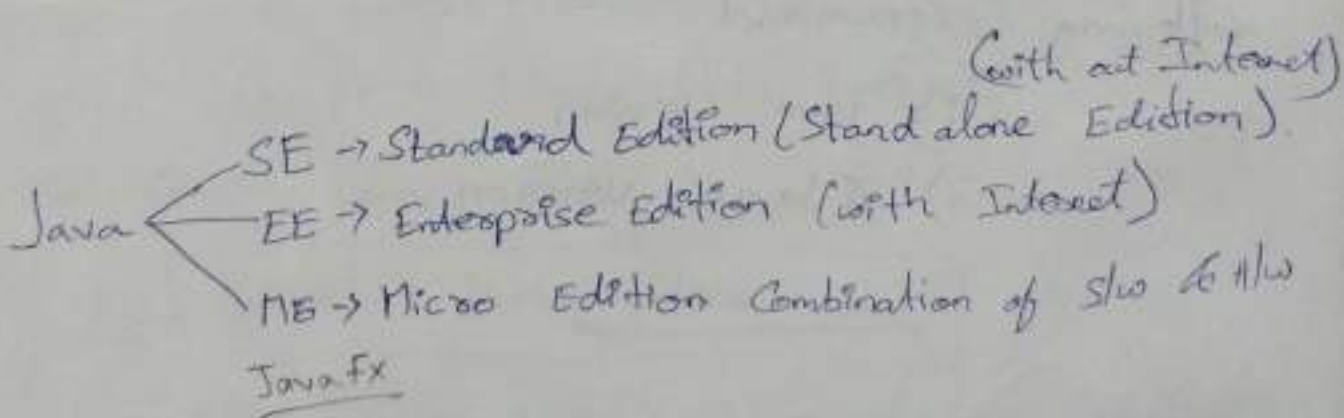
Company - Sun MicroSystems (Oracle)

Father of Java - James Gosling

Year - 1995 (First Edition version) JDK 1.0

Recent Version - Java 20

Learning Version - Java 8



Steps for Installation of Java

Step 1 :- Download and Instal JDK from Osade website

Step 2 :- Configure → Environment Variables

→ Search Edit Environment Variable

→ in System Variables Search for

Path

→ Click on "Edit"

→ Click on "New" and paste the Path.

C:\Program Files\Java\Jdk-11.0.16.1\bin.

→ Click 'ok' after Setting the path

Step 3:- Verify the Installation

→ open Command Prompt and use the following Command

i) Java -Version

ii) Javac -Version

D:
cd -> change Directory

1.) class FirstProgram
{
public static void main (String[] args) {
System.out.println("Hello! Wellcome to Java
Programming... Happy Learning");
}
}

o/p Hello! Wellcome to Java Programming... Happy Learning

→ Save the java file with .java Extention
→ open Command prompt and Navigate to the folder where the java file is Present

Eg:-
C:\Users\Nandeesh>D:
D:\>cd Nandeesh\java\CFCM12

→ To Compile java file

Syntax -> javac filename.java

Eg:- D:\Nandeesh\java\CFCM12>javac FirstProgram.java

→ To Run java file

Syntax -> java classname

Eg:-D:\Nandeesh\java\CFCM12>java FirstProgram

2.)
```
class SecondProgram {
public static void main (String[] args) {

System.out.println ("* * * * * * * * * *");
System.out.println ("Let begin and be focused in
                    learning");
System.out.println ("Enjoy Learning");
System.out.println ("* * * * * * * * * *");

}
}
```

o/p

```
* * * * * * * * * *
Let begin and be focused in learning
Enjoy Learning
* * * * * * * * * *
```

Eg:- Suppose you saving Program as Programs

```
javac Programs.java
java Program
```

It won't Run because

in java while Execution

we have to give the classname we entered for eg:-

```
javac Programs.java
java SecondProgram
```

(Considering the above program.

Commands

Single line Command	Multi line Command
// ———	/* —— —— —— */

3. class Program2 {
public static void main (String[] args) {
System.out.println("Hello"); // for String [" "] double Quotes
System.out.println(10); // for number character no need of Quotes
System.out.println('B'); // for Single character we used Single Quote.
System.out.println(20+20); // Act as addition because of number.
System.out.println("20"+"20"); // once the number is given in double quotes. it going to consider as String
System.out.println("Number="+20); // Act as Concadination
System.out.println("Number="+20+20); // left to Right Rule
System.out.println("20+20+" is Number");
System.out.println("Number="+(20+20));
System.out.println("*********");
}
}

+ addition
+ Concadination

o/p
Hello
10
B
40
2020
Number=20
Number=2020
40 is Number
Number=40
****

4) class A {
public static void main (String[] args)
{
System.out.println ("Bhavi");
}
}
class B {
public static void main (String[] args)
{
System.out.println ("tha");
}
}
class C {
~~public static void~~ } ⇒ Saved as Program3
↑ filename

o/p
> javac Program3.java
> java A
Bhavi
> java B
tha
> java C
Main Method is ~~Miss~~ Not defined in class C
↓ error when we not defining the main method

→ In a java file we can Define N no of classes and the file name can be any name.
→ Once we Compile this file we will Get N no of .class files.
→ We can Run the class only if Contains Main Method with that class name;

Tokens:- -> Basic Building Blocks of Java Program.

keywords, Identifiers, Literals/Values

keywords

=> are Predefined / Reserved words

=> All keywords Must be written in Small letters

=> 52 keywords in Java (32 in C)

Identifiers

-> The Name Given by the programmer.

* class Name
* Variable Name
* Method Name -> a block of code which only runs when it is Called.

Rules => Identifiers can have AlphaNumeric characters

=> Identifiers should not have any Special characters (_, $),

=> Identifiers Should not Start with Numeric

=> We should not use keyword as Identifier

Guidlines G1:- Class Name Should Start with Capital letter

G2:- Variable or Method Name Start with Small letters

G3:- Camlin Conventions

Student Details

Literals / Values

Literals are the Values

Premitive Datatypes	Sizes	Default Values
boolean	1 bit	false
Char	16 bits	' ' → unicode → '\u0000'
byte -128 127	8 bits	'\0' → ASCII → Null Point
Short -32768 32767	16 bits	0
int -2147483648 to 2147483647	32 bits	0
long	64 bits	0
float	32 bits	0.0
double	64 bits	0.0

Variables

→ Variable are Name Given to Memory Location.
→ It is like Container
→ It is an Identifies Token

* Variables Declaration.
  eg:- int a, b, c, ...;
* Variables Initialization
  eg:- a = 10;
* Variable utilizations
  eg:- System.out.println(a);

eg:-

```
int b;
b = 28;
System.out.println(b);
```

5.)
```
class EmployeeDetails{
public static void main(String[] args){

System.out.println("*-*-*-*-*-");
// variable Declaration
int empId;
String empName;
double empSalary;
double empCTC;
float empExp;
long phNo;

// variable Initialization.
empId = 1234;
empName = "Siva";
empSalary = 5000.0;
empCTC = empSalary *12;
phNo = 9999999991;
empExp = 3.5f;

// variable utilization
System.out.println("Employee Id=" + empId);
                   ("Employee Name=" + empName);
                   ("Employee Salary=" + empSalary);
                   ("Employee CTC=" + empCTC);
                   ("Employee Experiance=" + empExp);
                   ("Employee PhoneNumber=" + phNo);

System.out.println("*-*-*-*-*-*-");}
}
```

dl

--*-*-*-*-

Employe Id = 1234

Employe Name = Siva

Employe Salary = 5000.0

Employee CTC = 60000

Employe Experience = 3.5

Employe PhNo = 9999999999

--*-*-*-*-

Assesment-I

To print Employee details
Student

```
Class StudentDetails {
public static void main (String[] args ){
System.out.println ("!-!-!-!-!-!");
```

// Variable Declaration

```
int clgID;
String stdName, Department;
String ClgName;
float CGPA;
long RegNo;
```

// Variable initialization

```
clgID = 101;
stdName = "Sivaiya";
Department = "CSE";
clgName = "PEC";
CGPA = 9.3f;
RegNo = 111419104009l;
```

Variable utilization

```
System.out.println("CollegName c.
    "    "    "("Name of Student =
    "    "    "("College ID Number =
    "    "    "("Persued Department="+
    "    "    "("University Registration Number.
    "    "    "("Overall CGPA =" + CGPA);
System.out.println("!-!-!-!-!-!-!");
}
}
```

output:-

```
!-!-!-!-!-!-!
PEC
Name of Student = Shivaiya
College ID Number =101
Persued Department = CSE
University Registration Number. =111419104009
Overall CGPA =9.3
!-!-!-!-!-!-!
```

1.)
```
class Program1{
public static void main (String[] args){
S.O.Pln("_1_1_1_1_1_");
int b=28; // declaration + Initialization
System.O.Pln("b Value = "+b);

b=05; // Reassigning
System.Out.println("Reassigned b value = "+b);

final double Pi = 3.14;
System.out.println("Pi Value is = "+Pi);
}
}
```

Output

```
_1_1_1_1_1_
b Value = 28
Reassigned b Value = 05
Pi Value is 3.14
```

Here, final => is a keyword which act as Constant and fix the value

-> We Can define Constants in java by using final keyword.

-> If any variable is final it is never possible to reassigned them

write a program to add two numbers

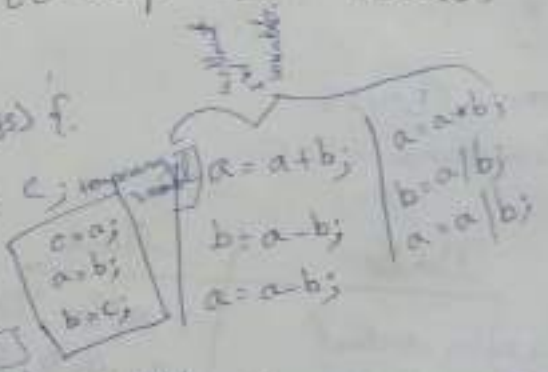
```
class Addition {
PSVM (String[] args) {
int a, b, c
a=28;
b=10;
c=a+b;
System.out.println("Addition of a&b"+c);
}
}
```

o/p

Addition of a&b = 38

write a program to swap two numbers

```
class Swap{
PSVM (String[] args) {
int a=10, b=20, c;
a=b;
b=c;
c=a;
System.out.println("After Swapping a&b"+a,b);
}
}
```



o/p

After Swapping a&b 20, 10

Write a program to find area and Circumference of circle.

Area πr^2

Circumference $2\pi r$

```
class circle{
psvm(String[] args){
int r=4;
double Pi=3.14;
float a=Pi*r*r;
float c=2*Pi*r;
System.out.println("Area="+a);
System.out.println("Circumferance="+c);
}
}
```

O/P

Area= 50.24
Circumferance = 17.12.

Double	float
→ After decimal 15 digits Can possible	→ After Decimal 7 digits Can possible
→ 64 bits	→ 32 bits

Assesment

```
Class Maths_1{
psvm(String[] args){
float a,b,c,d;
a= 3.2f;
b= 2.3 f;
c=((a+b)*(a+b));
d=((a*a)+(b*b)+(2*a*b));
System.out.println(" Algebraic Formula
    (a+b) power 2= a power 2+b power 2+2ab :-"
                +c+"="+d); }}
Class Maths_2{
Psvm(String[] args){
int a,b,c,d,e;
a=1;
b=2;
c=3;
d=(a*(b*c));
e=((a*b)*c);
System.out.println("Associative Formula a(bc)=(ab)c::-"
                +d+"="+e);}}
```

```
Class Maths-3{
PSVM(String[] args){
int r=2;
double h=2.2;
double v=(3.14*r*r*h);
System.out.println("Volume of Cylinder="+v);
}
}
```

$\pi r^2 h$

```
Class Maths_4{
PSVM (String[] args){
int b=1, h=9;
double triangle = 0.5*b*h;
System.out.println("Area of Triangle="+triangle);
}
}
```

$\frac{1}{2}bh$

```
Class Maths-5{
PSVM (String[] args){
int l=10, b=20, h=30;
int c=l*b*h;
System.out.println("Volume of Cuboid="+c);
}
}
```

lbh

output

Javac MathsPrograms.java

Java Maths_1

Algebraic Formula (a+b) power 2 = a power 2+b power 2+2ab ::-- 30.25 = 30.25

java Maths_2

Associative Formula a(bc) =(ab)c ::—6=6

java Maths_3

Volume of Cylinder = 27.632000000000005

java Maths_4

Area of Triangle =4.5

java Maths_5

Volume of Cuboid = 6000

Keywords (Token) (Predefined (or) Reserved keys)

i) keywords for data types are:

(1) boolean (2) byte (3) char (4) int (5) long
(6) Short (7) float (8) double

ii) keywords for access control are:-

(9) Private (10) Protected (11) Public

iii) keywords for modifiers are:

(12) abstract (13) final (14) native (15) private (16) Protected
(17) Public (18) static (19) transient (20) Synchronized

(21) Volatile (22) StrictFp

iv) Keywords for Catch-exception are:

(23) try (24) Catch (25) finally (26) throw

v) Keywords for loops as decision makers are:

(27) break (28) case (29) Continue (30) default (31) do
(32) while (34) for (35) Switch (36) if (37) if else

vi) Keywords for class functions are:-

(38) class (39) extends (40) Implements (41) Import
(42) instanceof (43) new (44) Package (45) return
(46) interface (47) this (48) throws (49) Void
(50) Super

vii) Keywords for assigned Values are.

(51) true (52) false (53) null

viii) Outdated Keywords are:-

(54) Const (55) goto

—— X —— X —— X ——

Datatype Varname ; // Variable declared. Declaration

~~Datatype~~ Varname = Var | Val | expression. // Variable ~~Declaration~~ initialization

Datatype Varname1 = Var | val | expression, Varname2 ... ... ;

Datatype. Varname = Var | Val | expression ; // Declaration

Scope of Variable

Local — Global in Nature

(inside the main method)

Static NonStatic.

(inside the class block)

Syntax

```
class Bhavika{
    // Global Variable
public SUM (S[] args) {
    // local Variable }
}
```

```
class xyz{                                  Return type
public  static  void  main (String[] args){
access specifier  keyword                   method
(or) modifier
if ( args.length == 0)
  {
   System.out.println(args.length);
  }  class Reference  method
  else {
    System.out.println("Error");
    }
  }
}
```

we can print without S.O.Pln(" ");

```
if (System.out.printf("Bhavi")){

}  //will Print the output
```

// search about it

Scope of Variables

```
{
  {
    {
    }
  }
}
```

```
{
int a=50;
// int a=100; error
  {
    S.O.Pln(a); //50
    int a=100;
  }S.O.Pln(a); //100
}
```

```
{
  {
    {
    }
  }
}
S.O.Pln(a); //50
```

In One Context we can't able to assign Same Variable name with Same data type.
its Shows errors.

```
Class Program2{
    Static int x=20;
    PSVM(String[] args){
    S.O.pln("Global x value = "+x); //20
    int x = 30;
    S.O.pln("local x value = "+x); //30
    S.O.pln("Global x value = "+Program2.x); //20
    }
}
```

O/P

Global x Value = 20
local x Value = 30
Global x Value = 20

Methods

=> A Set of Statements used to Perform task
=> We are Methods to achieve Code Reusability
=> Method Must Create in class Context

{ outside the class
outside the main }

In a class
=> Can Create 'N' number methods in Single Program but its executes will Starts from main method.

Syntax

(modifiers) Datatypes(or) void returntype methodname (arguments)

methodname ⇐ Identifier

(arguments) ⇒ int a
eg: int a, int b

{
    } set of Statements to perform
    return value;
}

// want to pass in individual datatypes

```
Class Addition {
    Static void Sum(int a, int b)
    {
        int c=a+b;
        System.o.println("Sum of "+a+" and "+b+" is "+c);
        return;
    }
    PSVM(String[] args){
        Sum(10,20);
        Sum(100,200);
        Sum(1000,2000);
    }
}
```

Repeats

} method can called "N" Number of times

O/P

Sum of 10 and 20 is 30
Sum of 100 and 200 is 300
Sum of 1000 and 2000 is 3000

Note for Return type

* In every Java method Return Statement is mandatory, If we are not writing the return Statement the Compiler will write the return Statement only in the Case of void return type

```
class Program{
static int/void cube(int x)
{
  int c = x*x*x;
  return c;
}
static int Square(int x)
  {
    int return x*x;
  }
psvm(String[] args){
  int x = cube(5);
  s.o.pln(x)
  s.o.pln(Square(3));
  s.o.pln(cube(5));
  }
```

o/p)

```
class circle
{
 static final double pi=3.14;
 static double area(int r)
 {
    return pi*r*r;
 }
 static double circum(int r)
  {
     return 2*pi*r;
 }
 static void Calculate(int r) {
    S.O.pln("Given radius = " + r);
    S.O.pln("Area = "+area(r));
    S.O.pln("Circumference ="+circum(r));
    S.O.pln("----------");
 }
 psvm(String[] args){
 System.O.pln("*********");

    Calculate(5);
     Calculate(10);
 System.O.pln("*********");

 }

}
```

o/p

* * * * * * * * * *

Given radius = 5

Area = 78.5

Circumference = 31.400000000000002

Given radius = 10

Area = 314.0

Circumference = 62.800000000000004

---

* * * * * * * * *

Method Overloading :-

i) Developing the Method with the Same Name with different argument list is known as Method overloading

ii) Arguments Should different in

a) length (0, 1, 2)

b) Datatype (0

c) Sequence

iii) Method Overloading used to achieve Complie time polymorphism.

```
class Program2
{
static void test()
{
  System.out.println("I am in test()...");
}
static void test(int x){
  System.out.println("I am in test(int)...");
}
static void test(double x){
   S.O.Pln("I am in test(double)....");
}
PSVM(String[] args)
{
S.O.pln("_1_1_1_1_1_1_1");
test(); // 1st method
 (or)
test(5); // 2nd method
 (or)
test(10.5); // 3rd method.
S.O.pln("_1_1_1_1_1_1_1");
}
}
```

O/P

```
_1_1_1_1_1_1_1
I am in test()....  // 1st method
 (or)
I am in test(int)... // 2nd method
 (or)
I am in test(double).... // 3rd method
                          _1_1_1_1_1_1_1
```

M:) whenever we want to perform a same type of operation with different input arguments, we use method overloading

Real time Example

1.) class Method_2

```
{
static final double pi = 3.14;
static float af(float a, float b){
float c = ((a+b)*(a+b));
float d = ((a*a) + (b*b) + (2*a*b));
S.O.pln(" Algebraic Formula "+c+"= "+d);
return 0;
}
static int asf(int a, int b, int c){
int d=(a*(b*c));   int c = a+b
int e = ((a*b)*c);
S.O.pln(" Associative formula "+d+" = "+e);
return e;
}
static double at(int b, int h){
double t = 0.5*b*h;
S.O.pln(" Area of Triangle = "+t);
return t;
}
```

```
Static double vc (double r, double h){
double v = pi * r * r * h;
S.o.pln("Volume of cylinder = "+v);
return v;
}
Static long cuboid (long l, long b, long h){
long c = l * b * h;
System.o.pln(" Volume of Cuboid = "+c);
return c;
}
PSVm (String [] args)
{
S.o.pln(" *_*_*_*_*_* ");
af (3.2f, 2.3f);
asf (1, 2, 3);
at (1, 2);
vc (4.3, 3.4);
Cuboid (235l, 345l, 8765l);
S.o.pln(" *-*-*-*-*-* ");
}
}
```

```
df
*_*_*_*_*_*
Algebraic formula 30.25 = 30.25
Associative formula 6 = 6
Area of Triangle = 1.0
Volume of Cylinder = 197.39924
Volume of cuboid = 71062375
```

Method overloading examples

1.) class RailwayTicketReservation

```
{
static void passenger (String name)
{
  S.O.pln (" Passenger Name -- " + name);
}
static void Passenger (String Depart, String Going)
{
  S.O.pln (" Train Departing from -- " + Depart);
  S.O.pln (" Train Going To -- " + Going);
}
Static void Passenger (double time1, double time2)
{
  S.O.pln (" Train Arrival time in Chennai = " + time1);
  S.O.pln (" Train Reaching time to Chennai = " + time2);
}
Static void Passenger (int fare, String s)
{
  S.O.P (" Ticket Cost from Chennai to Chennai park
          Station is " + fare);
  S.O.pln (" " + s);
}
PSVM (String[] args)
Passenger (" Bhavitha ");
Passenger (" Chennai ", " Chennai Park ");
    (12.35, 13.15);
    (10, " rupees ");
}
```

o/p

Passenger Name - Bhavika

Train Departing From -- Chrompet

Train Going To -- Chennai Park

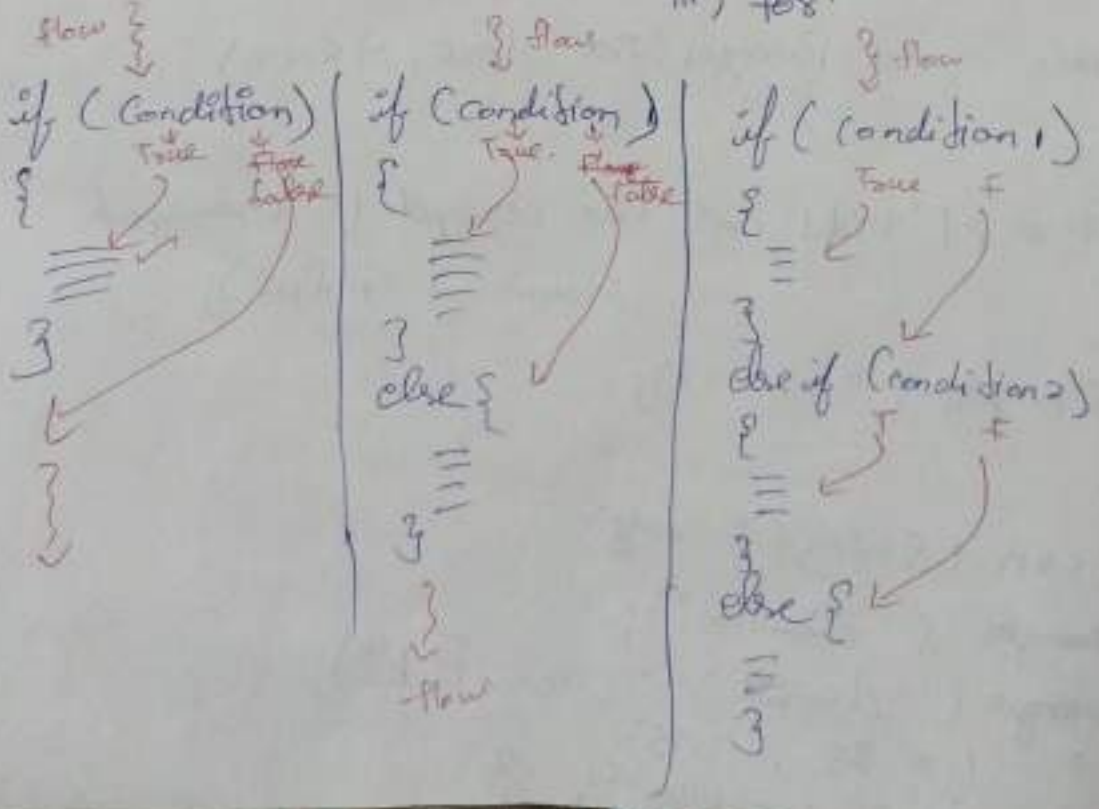
Train Starting time from Chrompet = 12.25

Train Reaching time to Chennai Park = 13.15

Ticket Cost from Chrompet to Chennai Park Station 10 rupees

Control Statement

- Decision Statements
  - i) if
  - ii) Switch
- Loop Statements
  - i) do
  - ii) while
  - iii) for



If Statement

```
Class Ifstatement{
psvm(String[] args){
  int age = 18;
  if (age >=18) -> do verify once.
   {
    s.o.pln (" Eligible for voating ");
  }
  else if (age <18){
     S.O.pln ("No Eligible for voting");
  }else{
     S.O. pln ("Not Interested for voting);
  }
}
}
```

Output

Eligible for Voting

write a program to check the Given number is even or odd

```
class check{
psvm(String[] args) {
int a = 8;
if ( a%2 ==0)
  {
  s.o. pln("A is Even number");
  }else { s.o.pln (" A is odd number"); }
  }
}
```

```
class oddEven {
static void check(int num)
{
    if (num%2 == 0)
        System.out.println(num + " is even");
    else
        S.O.println(num + " is odd");
}
PSVM(String[] args) {
    check(28);
    check(27);
}
}
```

o/p

28 is even
27 is odd

---

operators

- Arithmetic (+, -, *, /, %)
- Logical (&&, ||, !)
- Relational (>, <, >=, <=, ==, !=)
- Assignment (=, +=, -=, *=, /=, %=)
- bit wise (^, &, |, >>, <<, >>>, <<<)

operands

unary, Binary, ternary
(-, ++, --) ; ↓

(Condition)? True:false

Ternary Operand type example

```
class EvenOdd {
  static void check (num){
    S.o.pln(num + "is " + (num%2==0)? "Even": "odd");
  }
  psvm (String[] args){
    check (48);
  }
}
```

o/p

48 is Even.

Write a program to find largest of two numbers.
class also with three numbers

```
class P4{
  static void largest(int a, int b){
    if (a>b)
      S.o.pln(a + "is largest");
    else
      S.o.pln (b + "is largest");
  }
  static void largest ( int a, int b, int c){
    if (a>b && a>c)
      S.o.pln (a + " is largest");
```

(a>b)? "a is largest": "b is largest"

```
else if (b>c)
    S.O.pln(b+" is largest");
else
    S.O.pln(c+" is largest");
}
Psvm(String[] args){
largest(28,05);
largest(50,70,90);
}
}
```

o/p

28 is largest
90 is largest

---

ternary condition for largest of three numbers

((a>b && a>c)? a:(b>c)? b:c)

(a>b && a>c)? a:((b>c)? b:c)

(a>b && a>c)? a:(b>c)? b:c

```
class ATM {
psvm (String[] args) {
  int Pin = 7454;
  if (Pin == 7450) {
      double accbal = 5000.0;
      double amt = 5000.0;
    if (amt <= accbal && amt > 0)
        {
         s.o.pln ("Withdrawing amt =" + amt);
         accbal = accBal - amt;
         s.o.pln (" Balance amt =" + accbal);
        }
    else {
         s.o.pln ("Insufficient bal... !!!");
        }

  }
  else
      {
       s.o.pln ("Invalid Pin number.");
      }

  }
}
```

or

Invalid Pin number.

Assignment 4 - Control Statements

```
Class Assignment {
  PSVM (String[] args){
  Int num = 40;
  if (num%3==0 && num%5==0)
  S.O.pln ("SpamSlam");
  else if (num%3==0)
     S.O.pln ("Thala");
  else if (num%5==0)
     S.O.pln ("Thalaphathy");
  else
     S.O.pln ("Not divisible by 3 or 5");
  }
}
```

o/p

Thalaphathy

WAP to check ~~even or odd~~ +ve or -ve.

```
Class check {
  ~~static~~ v
  PSVM (String[] args){
  // Int num;
    num = 5
  if (num<=0) &&
    S.O.pln (num "Positive Number");
  else if (num >0)
       S.O.p (num "is negative");
```

num>0 +ve

num<0 -ve

num == zero Neutral

(n>1)?p:(n<1)?n:n

```
else if (num == 0)
    sopln(Natural number);

else
    sopln("_|_|_");

}
}
```

Switch Statement Syntax

```
switch (expression)
{                  └→ Value (int, char, String)
  case Value 1: ==
                ==
                break;

  case Value 2: ==
                ==
                break;
    :
  case Value n: ==
                ==
                break;

  default: ==
           ==
}
```

Multi-way branch statements

→ executes one statement from multiple conditions

i) Value name should not be Same in each Cases

ii) "break" is mandatory to perform individual case

iii) "default" can be written anywhere in the Switch Statement if that case want to mention "break" after default

iii) Java is a Case Sensitive

```
class switch{
Psvm (String[] args){
char ch='P';
Switch(ch)
{
Case 'R': S.O.pln("Red");
    break;
Case 'b': S.O.pln('black");
    break;
Case 'p': S.O.pln("purple");
    break
Case 'g': S.O.pln("Green");
    break
default : S.O.pln("orange");
}
}
}
```

o/p

purple

Calculator Program

```
class calculator{
  static void add(int a, int b){
      S.O.pln(a+"+"+b+"="+(a+b));
  }
  static void sub(int a, int b){
      S.O.pln(a+"-"+b+"="+(a-b));}
  static void mul(int a, int b){
      S.O.pln(a+"*"+b+"="+(a*b));}
```

```
static void div(int a, int b){
    s.o.pln(a+"/"+b+"-"+(a/b));}
static void Calc(int a, int b, String op){
    Switch(op){
        Case "add" : add(a,b);
                     break;
        Case "Sub": Sub(a,b);
                    break;
        Case "mul": mul(a,b);
                    break;
        Case "div": div(a,b);
                    break;
        default: s.o.pln("Invalid operation");
    }}
Psvm(String[] args){
    Calc(10, 20, "add");
         (20, 10, "Sub")
         (20*10, "mul")
         (20, 10, "div")
         (20, 10, "mod");
}
```

o/p

10+20 = 30

20-10 = 10

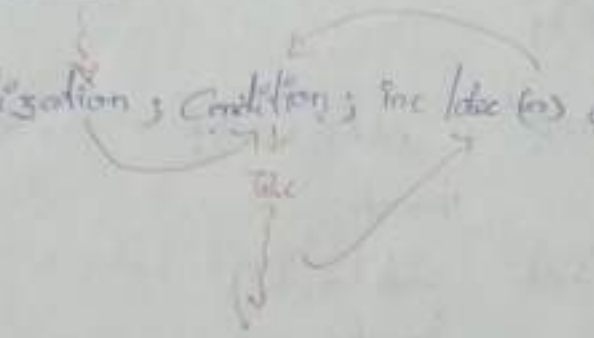
10*20 = 200

20/10 = 2

(Invalid operation)

Loop Based Control Statements

For loop Syntax



→ most Commonly used loop

⇒ Iterate a given set of statements multiple times.

```
class Programs{
    psvm(String a[] args){
        for (int i=7; i<=70; i=i+7)
        {
            S.o.println(i);
        }
    }
}
```

```
// for (int i = 1; i<=5 ; i++ ) 1 2 3 4 5
// for (int i = 1; i<=10; i++) 1 2 3 4 5 6 7 8 9 10
// for (int i = 0; i<=10; i=i+2) 0 2 4 6 8 10
// for (int i = 1; i<=10; i = i+2) 1 3 5 7 9
// for (int i = 1; i<=10; i = i+1) 1 2 3 4 5 6 7 8 9 10
```

o/p

```
7     56
14    63
21    70
28
35
42
49
```

Post Increment

x++

Pre Increment

++x

Eg:- int x=10

(i) x++ (10) + x++ (11) => 21

(ii) ++x (11) + ++x (12) => 23

(iii) x++ (10) + ++x (12) => 22.

=> Post Increment will fetch and then Increment the value

=> Pre Increment will Increment first and then it will fetch the value

(i) x++ (10) + x++ (11)

21

10 → 11

(ii) ++x (11) + ++x (12)

23

10 → 11 → 12

(iii) x++ (10) + ++x (12)

10 => 11 => 12

WAP to print 10 to 1

```
class for{
    psvm (String[] args){
    for (int i = 10 ; i <=10; i=i-1)
        { for (int int =10; i>=1; j--)
        {
        s.o.pln (i)
    }
}
}
```

for (i=1; i<=10; i++)
for(j=10; j>=1; j--)

WAP To check the Given number is Prime or not

```
class check{
    psvm (String[] args) {
        int num = 5;
        int count=0; 1 2
        for ( int i=1; i<=num; i++)
        {
            if (num%o.i ==0) {
                count++;
            }
            if (count>2)
            break;
        }
    if (count ==2)
        s.o.pln (num+ " is prime");
    else
        s.o.pln (num + " is not a prime");
}
}
```

o/p
5 is prime

Code Optimization

=> WAP to find the Sum of numbers

```
class find{
    psvm(String[] args){
    int n=10, sum=0;
    for(int i=1; i<=10; i++)
        {
        sum = sum + i;
        }
    s.o.pln(sum);
    }
}
```

o/p

55

=> WAP to find the factorial of Given number

```
class find{
    psvm(String[] args){
    int n=5, fact=1;
    for(int i=1; i<=n; i++)
        {
        fact = fact * i;
        }
    s.o.pln(fact);
    }
}
```

o/p

120

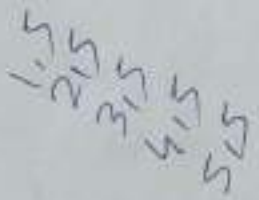
~~int n=5~~
~~int fa~~
~~for(int~~ int num = 5;
int fact = 1;

for (int i = 5; i <= num; i--)
{ for(int i=n; i>=1; i--)

fact = fact * i;

}
sopln (fact)

5*5
4*4



Class PatternProgram {

psvm (String[] args) {

for (int i=1; i<=5; i++)
{
for(int j=1; j<=5; j++)
{
s.o.pba(" * ");
}

s.o.pln(); }

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

```
int k=1;
for(int i=1; i<=4; i++)
  {
    for(int j=1; j<=i; j++)
      {
        S.O.P(k);
        k++;
      }
    S.O.Pln();
  }
```

1
2 3
4 5 6
7 8 9

---

class Pattern2 {

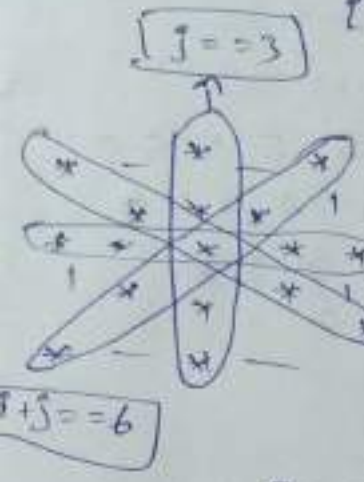
~~S.O.Pln~~

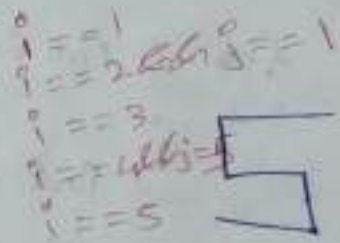
Psvm(String[] args){

```
for(int i=1; i<=5; i++)
  {
    for(int j=1; j<=5; j++)
      {
        if(i==1 || j==1 || i==5 || j==5)
          S.O.P("* ");
        else
          S.O.P("  ");
      }
  }
S.O.Pln("---------");
```

* * * * *
* *
* *
* *
* * * * *

```
for(int i=1 ; i<=5; i++)
{
  for( j=1; j<=5; i++)
  {
    if (i==j || i+j==6 || i==3 || j==3)
    {
      S.o.p("* ");
    }
    else
      S.o.p("  ");
  }
}
```





j=1 j=2 j=3 j=4 j=5

i=1
i=2
i=3
i=4
i=5

 => i==1 || i==5 || j==5 || j==2

 => i==1 || i==5 || j==3

 => i==1 || j==1 || i==3 || i==2 && j==5

 => j==5 || j==1 || i==j

 => j==1 || i==1 || j==5 || i==3

```
for( int i=1 ; i <=5 ; i++)
{
    int k=6
    for (int j=5; j>=i ; j--)
    {
        S.O.p(k-i);
    }
}
```

=>
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1

S.O.p(k-j) =>
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```
for ( int i=5; i>=1 ; i--)
{
    int k=6
    for(int j=1 ; j<=i ; j++){
        S.O.p(k-j)
    }
}
S.O.pln()
```

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

```
class Pattern{
  psvm(String[] args)
  {
    int k=5;
    for(int i=1;i<=5;i++)
    {
      for(int j=5; j>=i; j--)
      {
        s.o.pln("* ");
        s.o.pln(i+" ");
        s.o.pln(j+" ");
        s.o.pln(k+" ");
      }
      k--;
      s.o.pln();
    }

    for(int i=1; i<=5; i++)
    { int x=1;
      for(int j=5; j>=i; j--)
      {
        s.o.pln(x+" ");
        x++;
      } s.o.pln();
    }
  }
}
```

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

1 2 3 4 5
6 7 8 9
10 11
12 13
14
15

```
class Pattern{
  psvm (String[] args){
    for(int i=1 ; i<=5; i++){
      for (int s=5; s>i; s--){
        S.O.pl(" ");
      }
      for (int j=1 ; j<=i; j++){
        S.O.p("* ");
      }
    }
    for (int i=1; i<=5 ; i++){
      for (int s=5; s>i ; s--){
        S.O.p(" ");
      }
      for(int j=1; j<=2i-1; j++){
        S.O.p("* ");
      }
    }
  }
}
```

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

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

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

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

```
for (int j = 1 ; j <= 5 ; j++)
    S.O.println(j+" ");
for (int j = 5-1 ; j >= 1 ; j--)
    S.O.println(j+" ");
}
S.O.println();
```

```
1 1 4=5
5>=1
- - - - - 1
```

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

```
int k=5;
for(int i=1; i<=5; i++){
    for(int s=5; s>=i; s--){
        s.op(" ");}
    for(int j=1; j<=k; j++)
        s.o.p(j);
```

~~for(int j=k; j~~

```
    for(int j=k-1; j>=1; j--)
        s.o p(j)
```

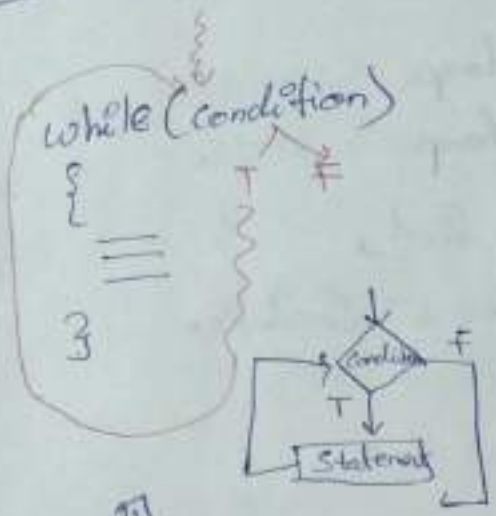
~~s.o.p(i)~~

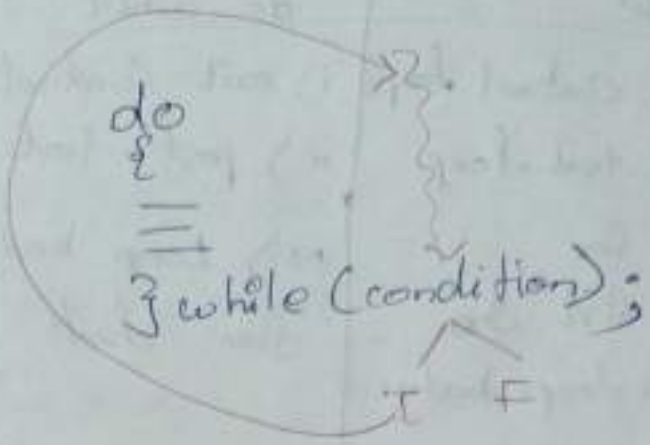
```
k
```

```
s=1   s<=i   s++
```

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

29 | 4 | 2023





while Syntax

a control flow statement that executes code repeatedly based on a given Boolean expression.

do while Syntax

used to execute a block of statements continuously until the given condition is true.

```
class while {
    psvm (String[] args) {
        int n = 1;
        while (n<=5) {
            s.o.pln("n value = "+n);
        } n++;
        s.o.pln("n value = "+n);
        do {
            s.o.pln("n value = "+n);
            n--;
        } while (n>0);
    }
}
```

o/p

```
n value = 1
"   "   = 2
"   "   = 3
"   "   = 4
"   "   = 5
after loop "   "   = 6
"   "   = 6
"   "   = 5
"   "   = 4
"   "   = 3
"   "   = 2
"   "   = 1
```

while	do while
i) entry Control loop	i) exit Control loop
ii) Pre-test loop	ii) post-test loop
iii) check for Condition first then run the loop body	iii) Loop body first, then check for a Condition

WAP to print the number in reverse order

```
class reverse{
 psvm(String args[])
  int n = 543;
  while (n > 0)
  {
  int rem = n % 10;
    s.o.p(rem);
    n = n / 10;
  }
```

543

54
10 | 543
   | 540
   3

5
10 | 54
   50
   4

0
10 | 5
   0
   5

3 4 5

Program for sum of digits

```
int n = 356, sum=0;
while (n>0)
{
int rem = n % 10;
 sum = sum + rem;
 n = n / 10;
}
s.o.pln (sum);
```

o/p
14

WAP to find the product of digits

```
int n=325, Prod=1;
while (n>0)
{
int rem = n%10;
  Prod = prod * rem;
   n = n/10;
}
S.O.pln (" Product of Given Number "+Prod);
```

```
105
5*2
10*3
30
```

O/P => 30

WAP to check the Given number is spy number.

```
int n = 123, Prod=1, Sum=0;
while (n>0)
{
  int rem = n%0;
  Prod = Prod * rem;
  Sum = Sum + rem;
  n = n / 10;
}
if ( Prod == Sum)
    S.o.pln ("it is Spy Number");
else
    S.O.pln (" it is not a Spy Number");
```

(when the sum of digits and product of digits are equal to each other is known as Spy number)

Printing Spy number b/w 1 to 100

```
for(int i=1; i<=100; i++)
{
  int n=i, sum=0, prod=1;
  while(n!=0){
  int rem= n%10;
    sum = sum + rem;
    prod = prod * rem;
    n = n / 10;
  }
  if (sum == prod)
    System.out.println(i);
}
```

1
2
3
4
5
6
7
8
9
22

WAP to check the given number is Armstrong or not

```
int n=153, sum=0;
while (n>0) {
int rem= n%10;
int cube= rem * rem * rem;
  sum = cube + sum;
  n = n/10;
}
if (sum == n)
  print ("Armstrong")
else
  Print ("it a Armstrong")
```

If the cube of each digits in a value is equal to the value

153
1 5 3
1 125 27
1 + 125 + 27

```
int n=153 , sum=0 , temp = n;
while n>0;
{
int rem = n%.10;
  sum = sum+rem*rem*rem;
n = n/10;
}
if (sum == temp)
    s.o.pln(" Armstrong");
else
    s.o.pln("Not a Armstrong");
```

WAP to check the Given number is Strong or not

Strong Number =>


```
class Strong
{
static int fact( int n)
  int fact = 1;
  for ( int i=1 ; i<=n ; i++)
      fact = fact * i;
      return fact;
}
psvm(String[] args).{ sum=0
int n=145 , temp = n;
while (n>0) {
   int rem = n%.10;
   sum = sum + fact(rem);
   n = n/10;
}
if (sum == temp)
  s.o.pln("Strong");
else
  s.o.pln("Not Strong");
}
```

1*1
1*2

Palindrome

```
int n = 121; int rev = 0, temp = num;
while (n>0)
{
int rem = n%10;
s.o.pln
rev = rev*10+rem;
n = n/10;
}
if (rev == temp)
  s.o.pln ("Palindrome");
else
  s.o.pln ("Not a Palindrome");
```

Fibonacci Series

```
int n=8; int a=0, b=1, c=0;
if (n==1)
{
s.o.pln(a);
}
else
{
s.o.pln (a+" ");
s.o.pln (b+" ");
n=n-2;
while (n>0){
  c = a+b;
  s.o.pln (c);
  a=b;
  b=c;
  n--;
}
}
```

0 1

—→ MALAYALAM ←

→ 121 ←

0 1

5

6-2

6>0

c = 0+1

1

0 1 1 2 3

4 > 0

Task with Big Number

WAP to count the number of digits in Given number

```
int n = 589, count = 0;
while (n>0)
{
int rem = n%10;
if
count++;
n = n/10;
}
```

WAP to print the Armstrong number between 1000 to

```
{
for(int i=1000; i<=10000; i++)
{
int n=i, sum=0; temp = n, a = n
int d=0
while(a>0)
{
d++;
a=a/10;
}
while(n>0) {
int r = r%10;
sum = sum + Power(r, d);
n = n/10;
}
if (temp == sum) System.out.println("Armstrong");
else
System.out.println("Not Armstrong");
}
```

```
static int power(int x, int d)
{
    int p = 1;
    for(int i=1; i<=d; i++)
    {
        p = p * x;
    }
    return p;
}
```

o/p

1634
8208
9474

WAP To print to check Palindrome

```
int n = 14341, rev=0, temp = n;
while(n>0){
    int x = n%10;
    rev = rev*10+x;
n = n/10
    ~~scanf(rev);~~
}
if (temp == rev)
    print("Palindrome")
```

WAP to print the numbers from 1 to 10 by using recursion

→ [Method Calling Itself is known as Recursion]

```
class B {
    static void print(int n)
    { if(n<=10){ sopln(n);
                 n++; print(n); }}
    psvm(String args[]){
        print(1);
    }
}
```

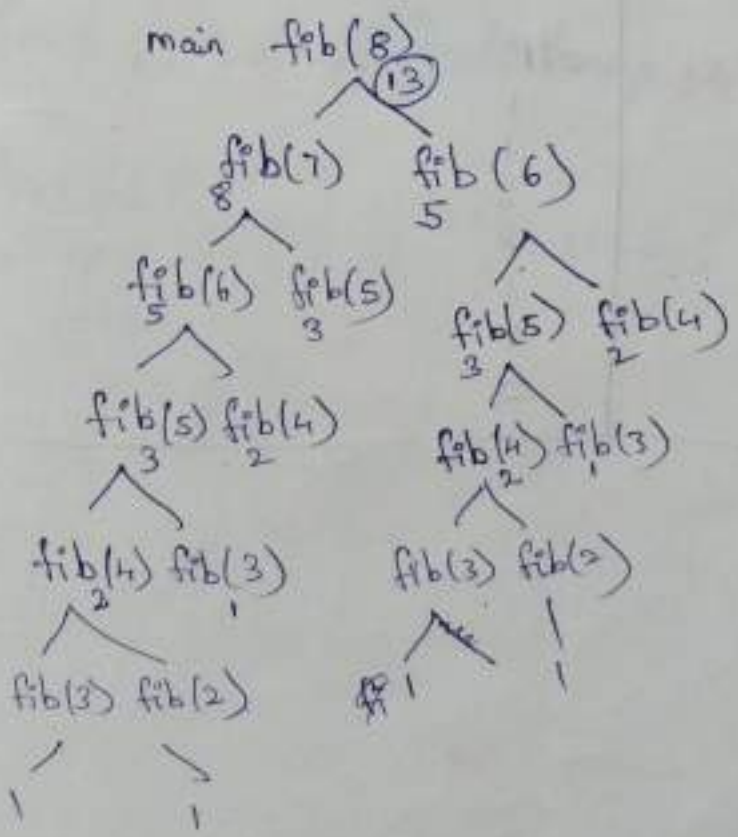
WAP to find the factorial of a number by using recursion.

```
class C {
    static int factorial(int n){
        if (n==0||n==1)
            return 1;
        else
            return n * factorial(n-1);
    }
    psvm(String args[]){
        int fact=factorial(5);
        s.o.pln(fact);
    }
}
```

f(5) => 5 * f(4) = 24
f(4) => 4 * f(3) = 6
f(3) => 3 * f(2) = 2
f(2) => 2 * f(1) = 1
f(1) => 1

WAP to find n^{th} Fibonacci number by using recursion.

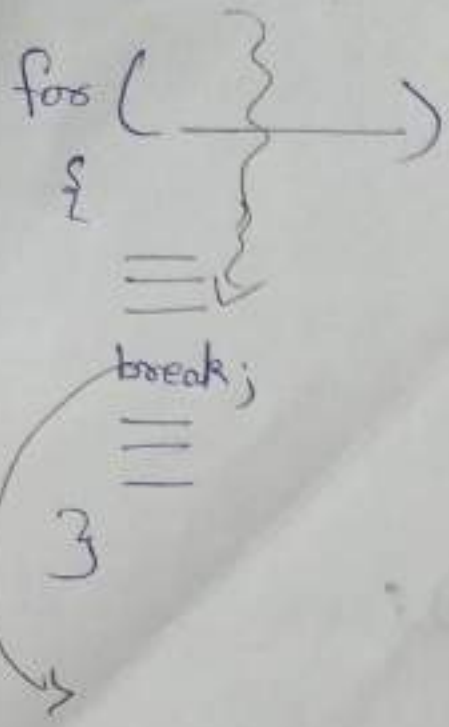
```
class A
{
    static int fib(int n){
      if (n==1) return 0;
    else if (n==2 || n==3) return 1;
    else return fib(n-1) + fib(n-2);

}
psvm(String[] args){
int x=fib(8)
  S.o.pln(x);
}
```



$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \quad \begin{matrix} 0 \\ 1 \\ 1 \end{matrix}$$

$n \times (n-1)$

break



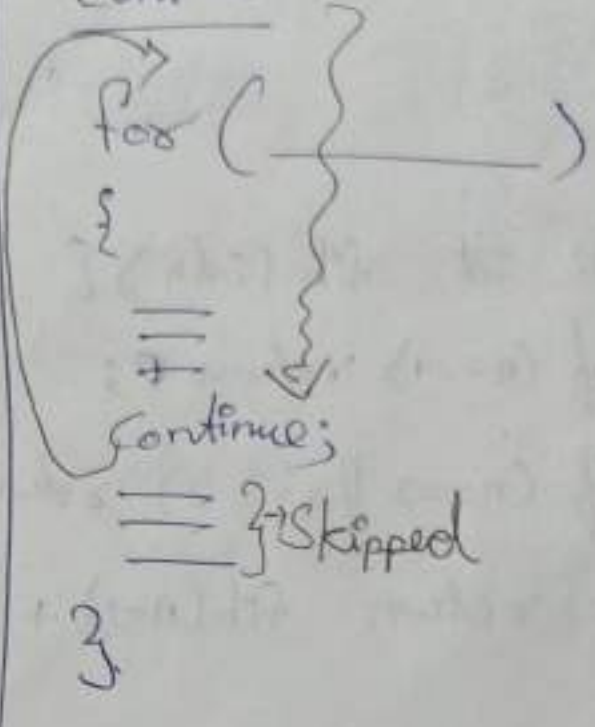
eg:-

```
for (int i=1; i<=5; i++){
    S.o.pln(i);
    if (i>=3)
        break;
    S.o.pln(i);
}
```

o/p

```
1
1
2
2
3
```

Continue



eg:-

```
for(int i=1; i<=5; i++)
    S.o.pln(i);
    if (i>=3)
        continue;
    S.o.pln(i);
```

o/p

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

```
for(int i=1 ; i<=5; i++)
 {
  ≡
 }
```

```
int i=1;
for( ; i<=5; i++)
 {
  ≡
 }
```

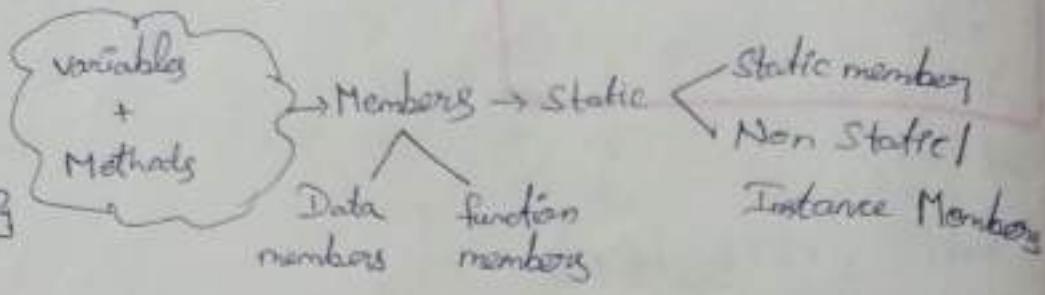
```
int i=1;
for( ; i<=5; )
 {
   i++;
 }
```

for(; ;) ⇔ Infinite Loop ⇔ while(true){ ≡ }

```
for(int i=1,j=1 ; i<=5, j<=5 ; i++, j++)
```

Object and class

Class className {



to access Static member

Static reference Syntax

```
className . staticmember
```

eg:- s.o.pln(Marker.ht)
Marker.write()

to access Non Static member.

Object reference Syntax

```
new className() . non-static member
```

eg:- s.o.pln (new Marker().price)
new Marker().draw()

eg)

```
class Marker
{
  Static int ht=10
      double price=200;
  Static void write()
      {
        =
      }
  void draw(){
        =
      }
```

```
class Marker{
    static int ht = 10;
        int price = 20;
    static void write(){
        s.o.pln("used to write");
    }
        void draw(){
        s.o.pln("used to draw");
    }
}
class Mainclass1{
    psvm(String[] args){
        s.o.pln("Marker Height=" + Marker.ht);
        Marker.write();
        s.o.pln("Marker Price=" + new Marker().price);
        new Marker().draw();
    }
}
```

Static reference Syntax

Non-static reference Syntax

o/p

Marker Height = 10

used to write.

Marker Price = 20

used to draw

```
class Marker{
    static int ht = 10;
        int price = 20;
    static void write(){
        s.o.pln("used to write");
    }
        void draw(){
        s.o.pln("used to draw");
    }
}
class Mainclass1{
    psvm(String[] args){
    s.o.pln("Marker Height=" + Marker.ht);
    Marker.write();
    s.o.pln("Marker Price=" + new Marker().price);
    new Marker().draw();
    }
}
```

Static reference Syntax

Static members

Non Static reference Syntax

non-static member

o/p

Marker Height = 10
used to write.
Marker Price = 20
used to draw

new classNew()

i) One new object is Created.

ii) It loads all not Static Methods.

⟹

```
class Prog1{
    static int x=10;
        int y=20;
        double z=33.33;
}
class Mainclass2{
    Psvm(String args[])
    s.o.pln("x Value = "+Prog1.x);
    Prog1.x = 100;
    s.o.pln("reassigned x Value = "+Prog1.x);
    s.o.pln("y value = "+ Prog1.y); new Prog1().y;
    s.o.pln("Z Value = "+ Prog2.z); new Prog1().z;
    new Program().y = 200;
    new Program().z = 333.33;
    s.o.pln(new prog1().x);
    s.o.pln(new prog1().z);
}
}
```

```
class Main class3 {
    psvm (String args[]) {
Prog1   P1 = new prog1();
    S.o.pln ("y Value = " + P1.y);
    S.o.pln ("Z Value = " + P1.z);
    P1.y = 200;
    P1.z = 333.33
    S.o.pln ("reassigned y Value = " + P1.y);
    S.o.pln ("reassigned z Value = " + P1.z);
prog1 (P2) = new prog1();
    S.o.pln ("y Value = " + P2.y);
    S.o.pln ("z Value = " + P2.z);
}
}
```

Assignment — Employee details

```
class nonstatic {
    Static String name = "Bhavitha";
        int empid = 101;
    Static String Company = "xxx";
}
class nonstatic1 {
    psvm (String args[]) {
    S.o.pln ("Name of Employee =" + nonstatic.name);
    S.o.pln ("Employe ID :" + new nonstatic().empid);
    nonstatic S1 = new nonstatic();
        S1.empid = 102;
```

```
s.o.pln("Reassigned Employee Id = "+ s1.empid);
s.o.pln("Company Name = " + nonstatic.company)
    }
}
```

o/p

Name of employee : Bhavitha
Employee ID = 101
Reassigned Employee Id = 102
Company Name = XXX

2)
```
class Pencil{
    static int ht=10;
      int Price = 5;
    void write(){
        s.o.pln("used to write");
    }
    void draw(){
        s.o.pln("used to draw");
    }
psvm(String args[]){
    System.out.println(Pencil.ht);
      Pencil.ht = 15;
    s.o.pln("reassigned = "+ Pencil.ht);
      Pencil c1 = new Pencil();
    s.o.pln(c1.price);
```

```
c1.price = 20;
S.O.println("reassigned cost = "+c1.price);
}
}
```

o/p

```
10
15 reassigned = 15
5
reassigned cost = 20
```

3)
```
class Gold {
    static int constantprice = 50,000;
           ct = 9;
}
psvm (String args[]) {
    s.o.pln ("Gold current Price = " + Gold.
                                constantprice );

    Gold.Constantprice = 45,000;
    s.o.pln (" Gold rate decreased = " + Gold.
                                constantprice );

    s.o.println(" Carrot of Gold = " + new Gold().ct);
    Gold ct1 = new Gold();
      ct1.ct = 8;
    G.o.pln (" Reduced Carrot of Gold = " + ct1.ct);
}
}
```

object and Class

object is a Real time entity which Contains Some States and behaviours.

States => properties of an object

behaviours => behaviour will represent the functionality of an object

---

Class is a Definition block which Defines States and behaviour of an object.

Class is like blue print of an object, if we have a class we can Create N number of objects

---

Without a class we can't Create objects

Inside the class we can define ~~N number~~ Variables (Data members) and Methods (function members) are Called as member of a class.

Their are two type of members Based on Static keyword

i) Static members

ii) NonStatic / Instance members

=> Static Members

i) They must be defined with Static keyword

ii) we can access static members by Static Refference Syntax that is

classname . staticmember

iii) Only one copy of stat Data in the Memory

=> ~~Non~~ Static Members

=> Non - Static Members

i) They should not be defined with Static keyword

ii) we can access non - Static members by creating object (object Refference Syntax) That is

new classname ( ) . non static members

iii) We will have multiple copies of non Static members.

Refference Variable

i) Referance Variable is a Special type of Varible which is used to store address of a object.

ii) Syntax

```
// Declaration
    classname ref1, ref2 ... ;
// initialization
    ref1 = new classname();
// utilization.
    ref1 . nonstatic member;
```

iii) The Default Value for Referance Variable is "NULL"

1.) If we want to access the member out side the class, Syntax must be follo

2.) If we want to access the members inside the class, Syntax is not mandatory but we should follow the following rules.

i) A static methods can access only with the static members directly

ii) A non static method can access both static and non static members.

ii) from the static method if we want to access non static method members we must create the object.

```
class A
{
static int x=10;
       int y=20;
static void m1();
    {
    S.o.pln(x); ✓
    S.o.pln(y); x
    }
    void m2();
    {
     S.o.pln(x); ✓
     S.o.pln(y); ✓
    }
P S V m ()
   {
     S.O.pln(x);
     m1();
     A a1 =new A();
     S.O.pln(a1.y);
   }a1.m2()
}
```

```
class B
{
   P S V m ()
   {
      S.o.pln(A.x)
      A.m1();
      A a1=new A();
      S.O.pln(a1.y);
      a1.m2();
   }
}
```

A b = new A()

2)

```
class Mainclass 3
{
  static int x=10;
       int y = 20;
  static void check()
    {
     s.o.pln ("*");
    }
        void test(){
      s.o.pln("#");
    }
psvm (S[] args)
  {
   s.o.pln (x)
  check();
  Mainclass3 ref =new Mainclass3();
   s.o.pln (ref.y);
   ref.test();
  }
}
```

Assignment

```
class epg
{
 static String name = "Bhavitha";
        int empId = 101;
 static String Company = "xxx";
```

```
Static int ht=10;
    int price=5;
Static void write(){
    S.o.pln("used to write");

}
    void draw(){
    S.o.pln("used to draw");
}
Static int Constantprice =50000;
    int ct=9;

PSVm (String[] args)
    {
  S.o.pln("- - - - - ");
  S.o.pln("Name of Employee="+epg.name);
  S.o.pln("Employee ID="+new epg().empid);
  epg b=new epg();
  b.empid=102;
  S.o.pln("Reassigned Employee ID="+b.empid);
  S.o.pln("Employee Company="+epg.Company);
  S.o.pln("- - - - - ");
  S.o.pln("Pencil height="+epg.ht);
  epg.ht=15;
  S.o.pln("pencil height changed="+epg.ht);
  S.o.pln("Price of pencil="new epg().price
  b.price=20;
  S.o.pln("Price of pencil Changed="+b.price);
```

```
write();
b.draw();
S.o.pln("_ _ _ _ _ _ _");
S.o.pln("Gold Current price = "+epg.Constantpr
  epg.Constantprice=45000;
S.o.pln("Gold Current price decreased="+
                        epg.Constantprice);
S.o.pln("present Gold Carrot ="+new epg();
b.ct=8;
S.o.pln("later Gold Carrot = "b.ct);
S.o.pln("_ _ _ _ _ _ _");
}}
```

```
int n=2;
for(int i=1; i<=n; i++)
{
    for(int s=1; s<=i; s++)
        {
          S.o.pln("*  ");
        }
    for(int j=1; j<=5-i; j++)
        {
          S.o.pln("#  ");
        }
  S.o.pln( );
    }
```

12/8/2023

Task of JVM

1.) It Allocates Memory
2.) It will Call class Loader
3.) It will Calls main for Execution.
4.) It will Calls Garbage Collector

1) It allocates Memory

-> static int a=10
e.p(a)

=> RAM is a valatile and only work with the power ~~Soft~~ Supply

=> ROM is a Non Valatile and it Can work with out power Supply.

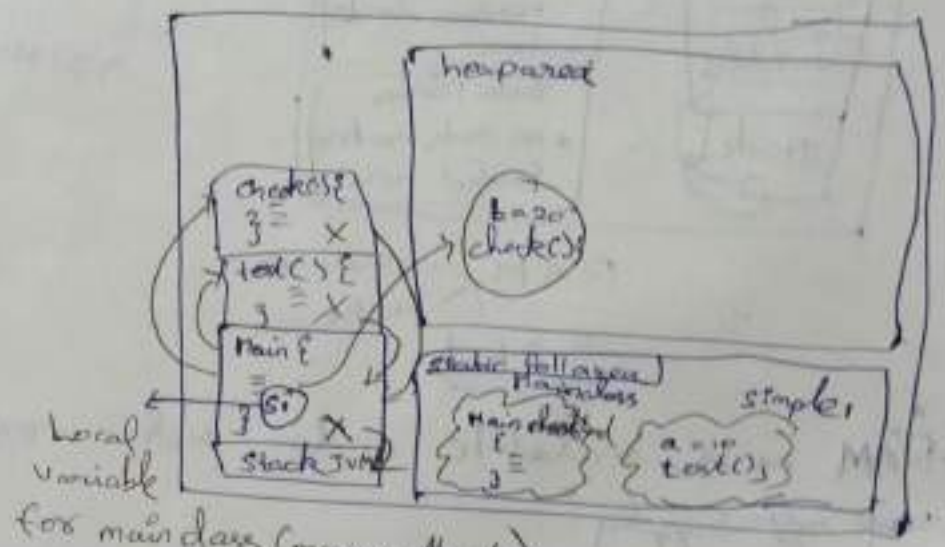
Task of JVM

```
class Simple1{
static int a=10
    int b=20
static void test(){
    s.o.pln("Tetit");
}
void check(){
    s.o.pln("checkit");
}
```

```
class Main class{
psvm(String args[]){
 s.o.pln(Simple1.a);
 Simple1.test();
 s.o.pln(
 Simple1 s1=new Simple1();
 s.o.pln(s1.b);
 s1.check();
}
}
```

O/P

10
Tetit
20
checkit



⇒ JVM class call classloader and it Pass to Main for execution at last it will be collected by Garbage Collector.

Static-Members

-> It will be loaded in the Static Poll Area
-> Static Members will be loaded at the time of class loading
⇒ A class loader will performs class loading only one time for a class

⇒ Hence, we will have Single copy of Static members

Non-Static Members

=> It will be Stored or loaded in the heap area

=> The Non-Static members will be loaded at the time of object Creation

=> We can Create "N" Number of objects. Hence, we will have multiple Copies of Non Static members.

Garbage Collection

⇒ It is a Process of Removing Dereffferanced objects or unreachable from the Heap area

=> The object Present in the Heap area without any reffference variable is known a dereffferand object, Their is no use of object and it will Consumes Heap memory

=> The Garbage Collector Can be Called Implicitly by the JVM at the end of the Program

⇒ As a programmer we Can Call GC Explicitly by using [System.gc();]

⇒ System.gc();
↓ built in class
↳ static method of System class

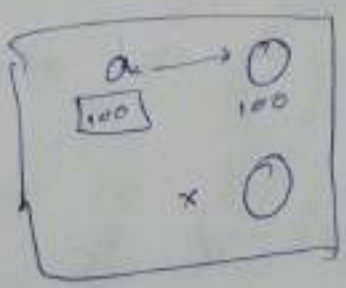
⇒ When we call the GC explicitly, It will internally Calls finalise method

⇒ The GC uses mark and sweep Algorithm

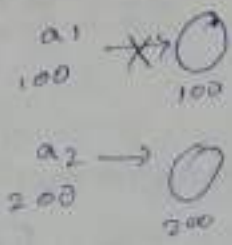
⇒ How to make the refferenced Variable ~~to~~ to get eligible for GC is we can make our object eligible for GC by removing refference variable, we can acheive this by assigning the null value.

```
class A
{

}
```

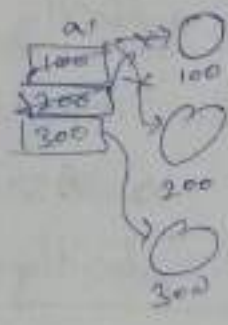
i)
```
A a = new A();
new();
System.gc();
```
1



ii)
```
A a1 = new A();
A a2 = new A();
a1 = null;
System.gc();
```
1

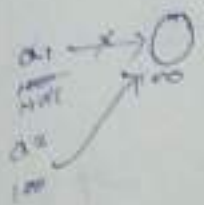


iii)
```
A a1 = new A();
a1 = new A();
a1 = new A();
System.gc();
```



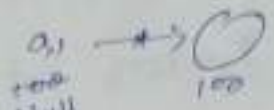
iv)
```
A a1 = new A();
A a2 = a1;
a1 = null;
System.gc();
```

1

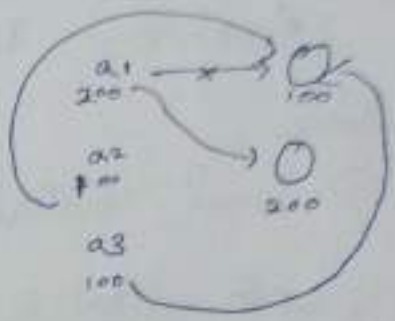


v)
```
A a1 = new A();
a1 = null;
A a2 = a1;
System.gc();
```

2



vi)
```
A a1 = new A();
A a2 = new A();
A a3 = a1;
a1 = a2;
a2 = a3;
System.gc();
```



vii)
```
A a1 = new A();
A a2 = new A();
A a3 = a1;
a1 = null;
a1 = a2;
a2 = a3;
System.gc();
```

viii)
```
A a1 = new A();
A a2 = new A();
a1 = null;
A a3 = a1;
~~a1 = null;~~
a1 = a2;
a2 = a3;
System.gc();
```

i) class to object
ii) S ∈ N-S
iii) Jar of JVM
iv) what is GC

⇒ Inside the class if we define variables or methods then they called as Members

⇒ Inside a method if we define variables it is not a Data Member but it is a local variable

⇒ Local variables are neither Static nor non Static

Types of Variable

→ Static Variables
→ Non Static Variables
→ local Variables

⇒ When to define the Variable as Static

i) when we want single copy of a variable then we should define the variable as Static

⇒ when to define the Method as Static

If a Method is using Only Static and local Variables then define that method as Static

```
class A
{
static  int x = 10; // Static variable
        int y = 20; // Non Static Variable

Static  void m1() // it is Static since method is
                  using static and local variable
  {
    int z=30; // local variable.
    s.o.pln(x);
  }
      void m2() // It is non-static bcz it is
                  using non-static variable
        {
          s.o.pln(y);
        }
}
```

Blocks or Initializers.

```
class A {

    static int a;
           int b;

    static {
           a=100;
           s.o.pln (" it is a static and
                    STATIC INITIALISATION BLOCK");
    }
    {
    b=200;
    s.o.pln (" it is a non static and
                    INSTANCE INITIALISATION BLOCK");
    }

}
```

class MC {

psvm(String args[]) {

System.out.println(A.a);

System.out.println(A.a);

A a1 = new A();

System.out.println(a1.b);

System.out.println(a1.b);

A a2 = new A();

}

}

O/P

it is a static and STATIC INITIALISATION Block

100

100

it is a non static and INSTANCE INITIALISATION Block

200

it is a non static and INSTANCE INITIALISATION Block

200

---

ff

```
class MC{

    psvm(String args[]){
     so.pln(A.a);
     s.opln(A.a);

     A a1 = new A();
     s.o.pln(a1.b);
     s.o.pln(a1.b);
     A a2 = new A();
    }
}
```

o/p

it is a Static and STATIC INITIALISATION BLOCK

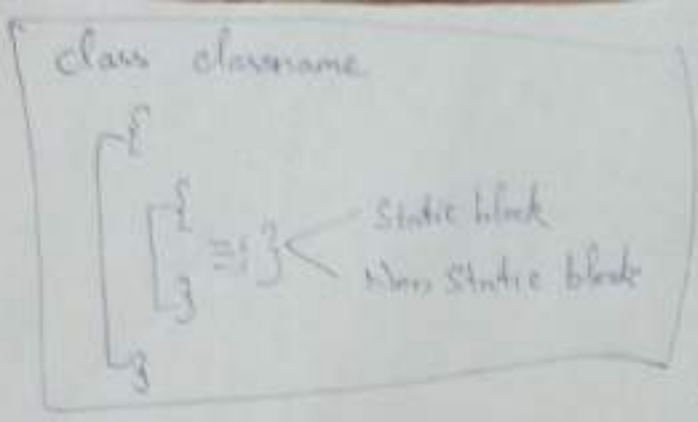
100
100

it is a non static and INSTANCE INITIALISATION Block

200

it is a non static and INSTANCE INITIALISATION Block

200



Syntax of block

DM
+
FM
+
blocks.

-> ~~blocks~~ can We Can Create blocks inside the class only and also can Create
-> N number of Blocks inside the class

Block:-

-> A set of statement enclosed within { } at class Level is known as Blocks

-> Java Supports two types of Blocks

i) Static Block
ii) Non-Static Block (or) Instance Block

Static Block => i) It must be defined with Static keyword

ii) It will executes only one time

iii) It will executes at the time of class loading

iv) It is used to initalize the static members of a class. Hence it is called as (Static Initialization Block)

Non - Static Block

i) It should not be defined with static keyword
ii) It will execute multiple times
iii) It will execute at the time of object Creation
iv) It is used to Initialize Instance Members of a class, Hence it is called a [Instance Initialization block]

NOTE:- The block are also called as Initializers
ii) Inside a class we can define "N" number of blocks but they will execute in Sequencial order.
iii) In a class if we have Static block, Non-Static block and Main Method the order execution will be.

Static block
Main Method
Non Static Block (only when we create an object)

Drawbacks of Blocks

1.) We can't call the Specific block explicitly
2.) We can't Pass any Arguments.

```
class A{
 static int x=10;
      int y=20;

 static {
      x=100;
      s.o.pln("block1.s");
    }
 static {
      x=1000;
      s.o.pln("block2.s");
  }
   {
   y=200;
   s.o.pln("block1.N.s");
   }
   {
   y=2000;
   s.o.pln("block2.N.s");
  }
}
class MC{
   static{
      s.o.pln("It is MC static");
   }
   {
   s.o.pln("In is MC non static");
   }
psvm{
    s.o.pln(A.x);
    s.o.pln(A.x);
```

```
A a1 = new A();
   S.o.pln(a1.Y);
A a2 = new A();
   S.o.pln(a2.Y);
MC m1 = new MC();

}
}
```

o/p

It is MC static

block 1.S
bbk 2.S

1000
1000
block1.NS
block1.NS
block2.NS

2000.NS
block2.NS
2000

It is MC non static

CONSTRUCTOR

Sample Program

```
class car{

    String Color;
    double Price;

Car (String arg1 , String arg2)
    {

        Color = arg1;
        Price = arg2;
    }
}
class Constructor {

PS.V.m(String[] args) {

    S.O.pln(" Creating and Initializing Car
                                    object");

    Car Car1 = new Car("Purple", 50000);
        S.O.pln (car1 . Color);
        S.O.pln (car1 . Price);
    Car  Car2 = new Car ("Blue", 60000);
        S.O.pln (car2 . Color);
        S.O.pln (car2 . price);

    }

}
```

old

Creating and initializing Car object

Purple
50000
Blue
60000

Constructor

class x {
DM
+
FM
+
blocks
+
Constructor
}

} Initialization methods

Constructor

i) It is a special type method where it is creating inside the ~~Constructor~~ class
ii) It will Create and Initialize the object
iii) Types of Constructor
i) zero Para (without Argument)
ii) Parameterized (with Argument)
iii) Default Constructor (will take default values for the Datatypes (Arguments)

=> Can Create 'N' number of Constructors in a class

iii) If we accept writing the Constructor the Compiler will create a default Constructor

iv)

```
classname ( int arg1, int arg2 ..... )
```

v)

classname Ref-Var = new classname (10, 20);

⇓

Constructor class

Constructor Overloading

Constructor Name

Having a Same ~~Con~~ Class Name with different argument list is known as Constructor Overloading

2.)

```
class Sample1
{
 int x;
 double y;
Sample1 ()
 { s.o.println(" Sample1 () ");
 }
Sample1 (int arg1)
    {
    s.o.pln (" Sample (int) ");
      x = arg1;
    }
Sample1 (int arg1, double arg2)
      {
      s.o.pln (" Sample (int, double) ");
        x = arg1; y = arg2; }
```

Ques MCI

```
class psum(s[] args)
  {
   Sample s1 = new Sample1();
    s.o.pln ("x" + s1.x)
    s.o.pln ("y" + s1.y);
   Sample s2 = new Sample(10)
     s.opln ("x" + s2.x);
     s.o.pln ("y" + s2.y);
    Sample1 s3 = new Sample1 (10,22.2);
       s.o.pln ("x" + s3.x);
       s.o.pln ("y" + s3.y);
  }
}
```

o/p)

```
Sample()
  x = 0
  y = 0.0
Sample(int)
  x = 10
  y = 0.0
Sample(int, double)
  x = 10
  y = 22.2
```

CONTRUCTOR

1.) It is a Special Member of class which is used to create and Initialize the objects

2.) It name must be same as classname.

3.) It should not have any Return type, but it Can take Arguments

4.)

SYNTAX

```
class classname
{
    classname(arguments)
    {
        == Initialization Logic
    }
}
```

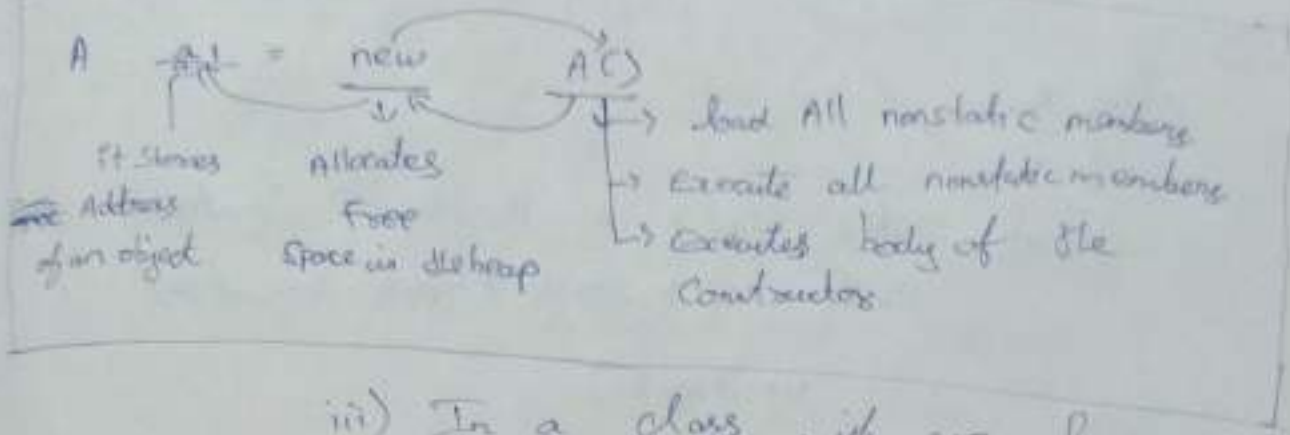
5.) For Every Java class Constructor is Mandatory, If we not writing the Constructor the Compiles will Defines Default Constructor

6.) As a Programmer, we can define Zero param Constructor (or) Parameterized Constructor.

7.) In a class we can Define 'N' number of Constructo but they should differ in arguments this is known as Constructor Overloading

NOTE : i) Constructor should not have any Returntype, If we write Returntype it becames Method.

ii) The Constructor should not be defined as Static it must be Non-Static



iii) In a class, if we have Block and Constructor, Block will executes fist after that Only Constructor will execute

iv) The Constructor will executes at the time of Object Creation Neither before nor after

---

CONSTANTS in Java

-> ~~There are 3 1.~~

=> We can Define Constants in Java by using final keyword

=> If we define any Varible as final we Can't reassign them

=> There are 3 types of Constants in java

i) Global Constant

Static variable with a final keyword

ii) object Constant

Non-Static Variable with final keyword.

iii) local Constant

local variable with final keyword

Q)

```
class A
{
final static int x=10; // Global Constant
    final int Y=20; // object Constant

        void m1()
        {
        final int z=30; //local constant
        }
```

```
class Employee
{
final static String company="ZOHO";
final int ID;
final String Name;
    double Salary;
Employee (int ID, String Name, double Salary)
{
    this.ID = ID;
    this.Name = Name;
    this.Salary = Salary;

}
Employee (int ID, String Name) {
        this.ID=ID;
        this.Name = Name; }
```

3 subject
Default Constructor
Average
Result
Pass
Fail

```
void display() {
    s.o.pln("Employee ID:" + ID);
    s.o.pln("Employee Name:" + Name);
    s.o.pln("Employee Salary" + Salary);

}
void cTC{
double
    return Salary;

}
}
```

```
class Employee Management System {

p
psvm (String args[]) {
    S.O.pln("Welcome to " + Employee.company);
    Employee emp1 = new Employee(101, "AAA", 20.0);
    Employee.display();
    emp1
    Employee emp2 = new Employee(102, "BBB", 3000);
    emp2.display();

}
}
```

O/P

```
Welcome to ZOHO
Employee ID:101
Employee Name: AAA
Employee Salary: 20.0
Employee ID = 102
Employee Name : BBB
Employee Salary : 0.0
```

```
class School{
  static final String SchoolName = "abc hr sec school";
  String StudentName;
  int StudentID;
  int Studentclass;
  static final String ExamName = "Mid term";
  double Tamilmark; T_m;
  double Telugumarks Te_m;
  double English; E_m;
School(String StudentName, int StudentID, int Studentclass,
       double T_m, double Te_m, double E_m);
    this.StudentName = StudentName;
    this.StudentID = StudentID;
    this.Studentclass = Studentclass;
    this.T_m = T_m;
    this.Te_m = Te_m;
    this.E_m = E_m;
}
void display(){
    S.o.pln("Student Name::" + StudentName);
    S.o.pln("Student ID ::" + StudentID);
    S.o.pln("Studentclass ::" + Studentclass + "th");
}
double total(){
    double sum = T_m + Te_m + E_m;
    S.o.pln("Total of 3 Subject ::" + Sum);
    return sum;}
double Avg{
    double avg1 = (T_m + Te_m + E_m) / 3;
    S.o.pln("Average of 3 Subject :" avg);
    return avg; }
```

```
void result(){
[if(Tm>=35 && Te>=35 && Em>35)
    S.O.pln(StudentName+" Pass in all Subjects");
 else S.O.pln(StudentName+"Fail in all Subjects")]
}
}
class SchoolMangementSystem{
    PSVM(String[] args)
{
 S.O.pln("----------");
S.O.pln(School.SchoolName);
School sd = new School("AAA",101,5,38.5,500,45.6);
sd.display();
 S.O.pln(School.ExamName);
 sd.total();
  Sd.Avg();
  sd.result();
S.O.pln(School.SchoolName);
School sd1 = new School("BBB",102,5,25.4,49.3,
                                        44.7);
 sd1 = display();
 S.O.pln(School.ExamName);
 sd1.total();
 sd1.Avg();
 sd1.result();
S.O.pln("----------");
```

abc hr sec school
Student Name :: AAA
Student ID :: 101
Student class :: 5 th
Mid term
Total of 3 Subject :: 134 1
Average of 3 Subject :: 44.6996
AAA Pass in all Subjects.

abc hr sec school
Student Name :: BBB
Student ID :: 102
Student class :: 5th
Mid term
Total of 3 Subject: 119.399999...
Average of 3 Subject: 39.8
Thala Fail in all Subjects.

Relationship
has-a / Composition
is-a Inheri

COMPOSITION

→ A object consisting of some other object is known as Composition (or)
→ A class which is defined with reference variable of some other class as its Data member is known as Composition.

⇒ Composition is also otherwise called as has-a type Relationship

⇒ Java Supports two types of Composition

1) Static Composition
2) Non-Static Composition (or) Instance Composition

1) Static Composition

* In Static Composition the ref var must be defined with static keyword
* We can access Static Composition by using Static Reference Syntax that is

classname . Static ref var . Non Static members;

[* The Static ref var will be created in the Static Pool area but the object is created in the heap Area]

* We will have Single Copy of Static Composition object

2) Non Static Compositions

* In Non-Static Composition the ref var should not be defined with Static keyword

→ Composition is also Called as has-a type Relationship

⇒ Java Supports two types of Composition

1) Static Composition
2) Non-Static Composition (or) Instance Composition

1) Static Composition

* In Static Composition the ref var must be defined with static keyword
* We can access Static Composition by using Static Reference Syntax that is

[classname.staticref var.Non Static members;]

[* the Static ref var will be loaded in the Static Pool area but the object is Created in the heap Area]

* We will have Single Copy of Static Composition object

2) Non Static Composition

* In Non-Static Composition the ref var should not be defined with Static keyword

⇒ Composition is also Called as has-a type Relationship

⇒ Java Supports two types of Composition

1) Static Composition
2) Non-Static Composition (or) Instance Composition

1) **Static Composition**

* In Static Composition the ref var must be defined with Static keyword
* We Can access Static Composition by using Static Reference Syntax that is

classname . Static ref var . Non Static members;

[* The Static ref var will be loaded in the Static Pool area but the object is Created in the heap Area]

* We will have Single Copy of Static Composition object

2) **Non Static Composition**

* In Non-Static Composition the ref var should not be defined with Static keyword

* We can access Non-Static Composition by Creating object

* For Each object Creation there a Seperate Non-Static Composition object will be Created in the Memory

NOTE:- Explain [System.out.println();]

⇒ It is a Good Example for Static Composition and Method Overloading

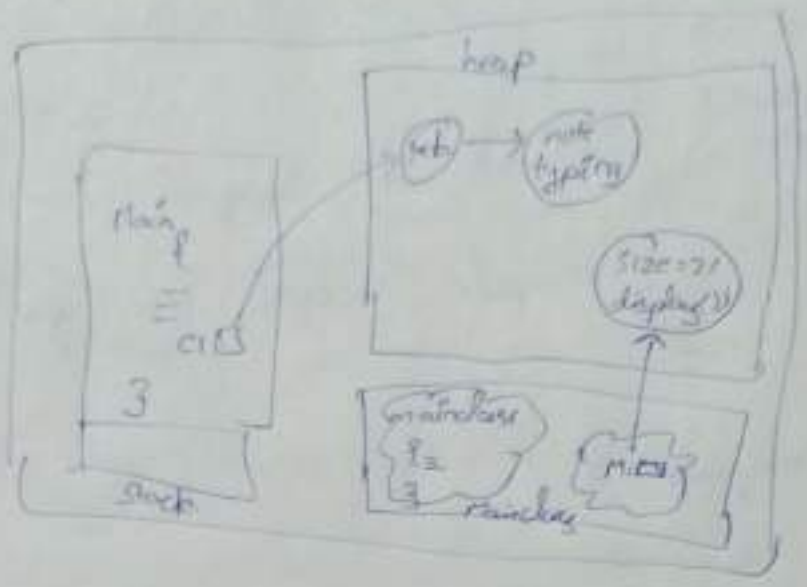
System.out.println();

- System → It is a built in class in Java
- out → Static ref var of PrintStream class present in System class
- println() → It is a non Static overloaded method of PrintStream

Composition Program

```
class Monitor {
    {
    int size = 21;
    void display() {
        S.o.println("Used to display");
    }
}
```

```
class KeyBoard {
    int nok = 104;
    void typing()
    {
        S.o.pln("used to type");
    }
}

class Computer {
{
static Monitor m1 = new Monitor();
    KeyBoard kb = new KeyBoard();
}
class MainClass1 {
PSVM(String[] args){
    S.o.pln("Size of the Monitor =" +
                Computer.m1.Size);
    Computer.m1.display();

    Computer c1 = new Computer();
    S.o.pln("Number of keys=" + c1.kb.nok);
    c1.Kb.typing();
}
}
```




```
class Engine
{
  int cc;
  Engine(int cc)
  {
    this.cc = cc;
  }
}
```

```
class Bike
{
  String name;
  Engine e;

  Bike(String name, Engine e)
  {
    this.name = name;
    this.e = e;
  }
}
```

```
class Mainclass {

  PSVM (String argsE]) {

  Bike rx = new Bike ("Rx-100", new Engine(125));
  s.o.pln ("Name of Rx bike" + rx.name);
  s.pln (" Engine cc of Rx = " + rx.e.cc);
  Bike ktm = new Bik ("KTM", new Engine(200));
  s.o.pln ("Name of bike" + KTM.name);
  s.o.pln ("Engine cc of bike = " + KTM.e.cc);
}}
```

if Name of bike = RX-100
Engine cc of RX = 100
Name of bike = KTM
Engine cc of KTM = 200

class Address {
int dno;
String Street;
String city;
int pin;
Address (int dno, String Street, String City, int pin) {
this.dno = dno;
this.Street = Street;
this.City = City;
this.Pin = Pin;
}

class Employee {
final static String Company = "Zoho"
final static ~~String~~ Address office = new Address
(34, "marigamaan Street", "Chennai",
600027)
final int id;
final String name;
double Salary;
Address residence;
Employee (int id, String name, double Salary, Address residence)
{ this.id = id;
this.name = name;
this.Salary = Salary; } }
this.residence = residence;

```
p class Main class {
    psvm (String args[]){
        s.o.pln (Employee.Company);
        S.o.pln (Employee.office.dono + ", " +
            Employee.office.Street + ", " +
            Employee.office.City + ", " + "Pin" +
            Employee.office.Pin);

Employee emp1 = new Employee (123, "Sharwin",
                50000.0, new Address(21,
                "zyx Street", "www", 6458);
        S.o.pln (emp1.id);
        S.o.pln (emp1.name);
        S.o.pln (emp1.Salary);
        S.o.pln ("Dono =" + emp1.residence.dono);
        s.o.pln ("Street=" + emp1.residence.Street);
        SO.pln ("City=" + emp1.residence.city);
        S.O.pln ("Pin=" + emp1.residence.Pin);
    }
}
```

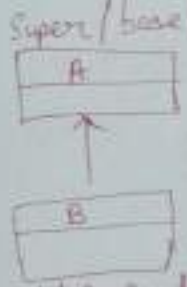
INHERITANCE

⇒ An object is acquiring the Properties of some other object is known as Inheritance

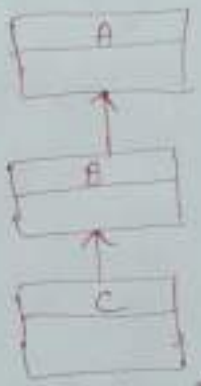
⇒ We use Inheritance to achieve

i) Code Reusability
ii) extensibility
iii) Constructor chaining

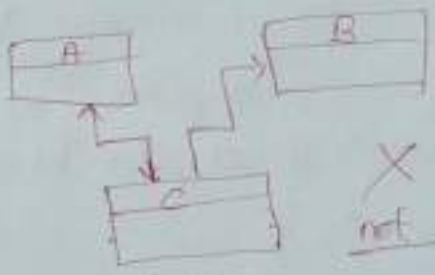
⇒ There are 5 types of Inheritance, but Java won't Support Multiple Inheritance



i) Single-Level Inheritance ✓

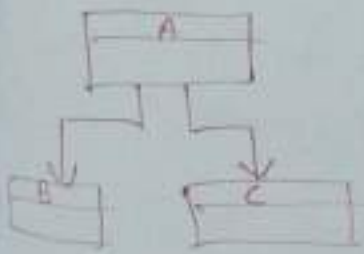


ii) Multi-Level Inheritance ✓



✗ not Possible

(iii) Multiple Inheritance.



iv) Hierarchical Inheritance

v) Hybrid Inheritance (Combination of other types)

i) Single Inheritance

```
class A {
    int x = 10;
}
class B extends A {
    int y = 15;
}
class m1 {
    psvm(String args[])
    {
    B b1 = new B();
    S.o.pln(b1.x);
    S.o.pln(b1.y);
    } }
```

extends keyword to Inheritance of other class Properties.

o/p
10
15

ii) Multilevel Inheritance.

```
class A {
    int x = 10; }
class B extends A {
    int y = 20; }
class C extends B {
    int z = 30; }
class m2 { psvm(String args[]) {
        C c1 = new C();
        S.o.pln(c1.x);
        S.o.pln(c1.y);
        S.o.pln(c1.z);
    }
}
```

o/p
10
20
30

Super Statement

```
class A {
    A(int x) {
        S.O.println("A Constructor");
    }
}
class B extends A {
    B() {
        super(10);
        S.O.println("B Constructor");
    }
}
class C extends B {
    C() {
        super(10.5);
        S.O.println("C Constructor");
    }
}
class MC {
    psvm(String args[]) {
        C c1 = new C();
    }
}
```

O/P

A Constructor
B Constructor
C Constructor

Def

What Java won't Support Multiple Inheritance?

1.) Syntax not Supported.

2.) Ambiguity of daimond Problem

3.) Multiple super() Statements not Allowed in Constructor

→ Let us assume two 3 classes where it is Internally Sub class of object class by the Compiler and also it has Properties which is extends beg to two classes and now as a third class it will get Confusion to get it from first or Second class so it is Called as Ambiguity and this problem will assumed as Dimond Shape. so it is knows as Ambiguity of daimond Problem

Method	Constructor
1) A Set of Instructions to Perform a task	It is Special type of members used to Create and Initialize the object.
2) Method can have a return type	Constructors will not have any return type, if we define return type it become method
3) we can define Method with Static keyword	we can't able to define Constructor with Static keyword
4) We Can pass No arguments	We can pass the arguments
5) Using Method is programmer choice	Using Constructors is also programmer choice but if it is not written means Compiler will take Default Constructor
6) Method can be created when we explicitly call it	Constructor get created only when object is Created
7) Method name will not have Same name as classname	Constructor name must be same Same name as class name.

this (Statements)

```
class A{
    A(){ this(10);
        S.o.pln("A()");
    }
    A(int x){ this(10,10);
        S.o.pln("A(int)");
    }
    A(int x, int y){
        S.o.pln("A(int,int)");
    }
    PSVM(String args[]){
        A a = new A()
    }
}
```

o/p
```
A(int,int)
A(int)
A()
```

Constructor Chaining

=> Calling a Constructor from any other Constructor is known as Constructor Chaining

=> We can achieve Chaining by Super();Statement (or) this();Statement

-> Super() must be written inside the first line Constructor as a

Super() Statement

i) Super() Statement is used to call Super class Constructor from Sub class Constructor

ii) To write Super Statement Inheritance is Mandatory

iii) Super() Statement can be written Implicitly by the Compiler or explicitly by the Programmer.

iv) If Compiler is writing the Super Statement we must have Zero Param, or Default Constructor in the Super() class

v) If there is no Zero param or default Constructor in the Super() class, we must Super() Statement explicitly by passing arguments

vi) To achieve Inheritance Super Statement is mandatory.

this() Statement

-> It is used to call Current or Same Class Constructor

-> this() statement should be written always explicitly

-> To write this statement Constructor overloading is mandatory || In a class if we have 'n' Constructors the maximum this statement we can write is "n-1". or else Recursive Constructor Invocation

Super and this keyword

```
class A { int x = 10; }
class B extends A {
    int x = 100;
    void b(){ int = 1000;
        S.o.pln(x);
        S.o.pln(this.x);
        S.o.pln(this.g.super.x);
    }
}
class Main(){
psvm(String args[]){
    B b1 = new B();
    b1.b();
}
}
```

O/P:
1000
100
10

→ Super keyword is used to access Immediate Super class members.

→ this keyword is used to access Current class members.

→ Super and this keyword can be written anywhere (except Static Context)

NOTE :- We can't Use Super and this keyword inside Main Method because it is static Method.

Super()	this()
→ Super() can call Super class Constructor	→ this() can call overloaded Constructor
→ Must be written inside Sub class Constructor as a first line.	→ Must be written inside Constructor as a first line
→ refer Superclass object	→ refer current instance of a class inside the Class

Method OverRidding

Program

```
class x{
    void Payment(){
        s.o.pln("Cash on Delivary");
    }
}
class y extends x{
    void Payment(){
        s.o.pln("Pay by using Card.");
    }
}
class z{
    psvm(String args[]){
        y a = new y();
        a.Payment();
    }
}
```

o/p

Pay by using Card.

why Static Method can't able to Inherit?

Method Overriding

=> Inheriting the method and changing its implementation in the Sub class According to the Sub class Specification is known as Method Overriding

=> To achieve Method Overriding Inheritance is mandatory

-> Method overriding is used to achieve Run time Polymorphism.

=> Whenever we want to perform Same type of operation in a different way in a Sub class, ~~According to the~~ we can use Method Overriding

-> while performing Methode overriding Should not change method Signature return type

Syntax

returntype methodname (Argumants) must be Same (But we can change body of the Method.

=> The Following Method can't be overrid

i) Private Instance
ii) Static Method
iii) Final Instance Method.

Modifier	Inheritance	Method overriding	overloading	
			Same class	Subclass
Private	x	x	✓	x
Static	x	x	✓	x
Final	✓	x	✓	✓

↓

it can be overriding only in Sub class

Diff b/w Method Overloading | Method Overriding

Method Shadowing

```
class A {  int x = 10
    static void (int x) {
        }   static int x=10;
}
class B {
    static void (int y) {
        int x = 10
    }
}
```

b/c x

Final Keyword

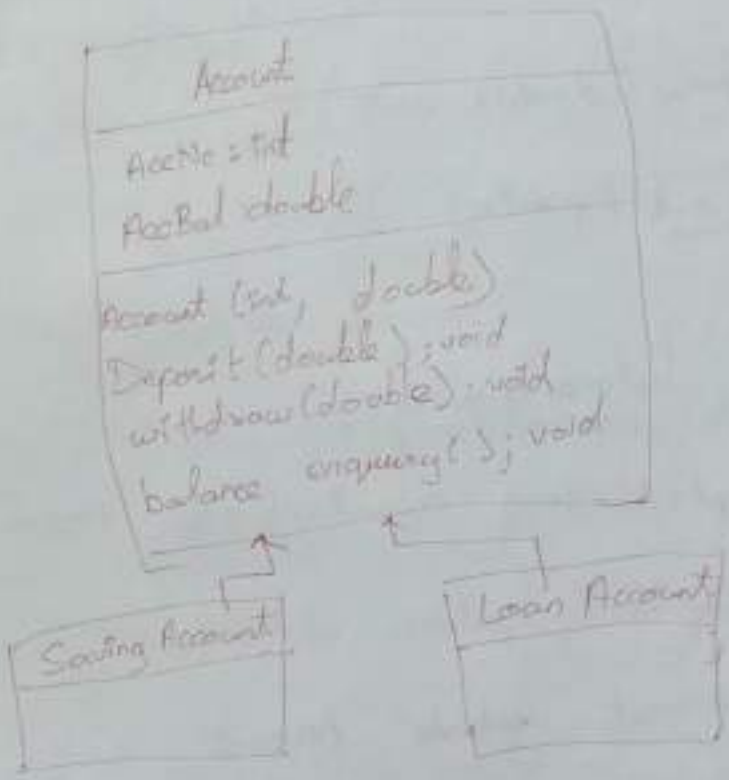
i) It is a keyword which is used to define constant

ii) If variable is final we can't reassign it

-> If a Method is final We can't Override it

-> If a Class is final we can't Inherit

[Static variable will not Inherit but can access it by Creating Object]




```
Package bankApp;
class Account
{
final Static String bankName = "MBank";
final int accno;
final String accholdername;
double accbal;
Account (int accNo, String accholdername, double accBal)
{   this . accno = accno;
    this . accbal = accbal;
    this . accholdername = accholdername;
```

```
System.out.println("Account Created Successfully !!!");
}
Account (int accno, String accholdername);
{
this (accno, accholdername, 0.0);
}
void deposit (double amt)
{
System.out.println("Dummy ...");
}
void withdraw (double amt);
{
System.out.println("Dummy ...");
}
final void balanceenquiry()
{
S.O.pln ("your account balance."+acbal);
}
}
class SavingsAccount extends Account
{
SavingsAccount (int accNo, String accholdername, double acbal)
{
super (accno, accholdername, acbal);
S.O.pln ("SavingsAccount Created Successfully...");
}
SavingsAccount (int accNo, String accholdername)
{
super (accno, accholdername);
S.O.pln ("SavingsAccount Created Successfully...");
}
```

```
void deposit (double amt)
 {
 s.o.pln(" Depositing amount in SavingsAccount:"amt);
 accbal = accbal + amt;
 balance = enquiry();
 }
 void withdraw (double amt)
   {
   s.o.pln(" Withdrawing amount from SavingsAccount:"amt);
   accbal = accbal - amt;
   balance enquiry();
   }
 }
class loanAccount extends Account
 {
 loanAccount (int accno, String accholdername, double accbal)
 {
 super(accno, accholdername, accbal);
 SOpln(" Loan Account Created Successfully");
 }
loanAccount (int accno, String accholdername)
 {
 Super (accno, accholdername);
 S.O.pln(" loan Account Created Successfully");
}
void deposit (double amt)
   {
   s.o.pln(" Deposit amt in LA-" +amt);
   accbal = accbal - amt;
   balance enquiry();
 }
```

```
void withdraw (double amt)
  {
  S.O.pln (" withdrawing amount from LA-"+amt);
  acc bal = acc bal + amt;
  balance enquiry ();
  }
Public class BankApp {
  public static void main (S[] args)
  {
S.O.pln ("welcome" + Account. bankname);
Savings Account Sal = new SavingsAccount (12345, "AAA", 5000.0);
Sal. balance enquiry.();
Sal. deposit (3000.0);
Sal. withdraw (1500.0);
Account Ia1 = new LoanAccount (31567, "xxx", 70000);
Ia1. balance enquiry ();
Ia1. deposit (25000.0);
Ia1. withdraw (3000.0);
}
}
```

Type Casting

- Primitive Casting
  - Widening
  - Narrowing
- Class / Reference casting
  - upcasting
  - downcasting

Widening => used to change Smaller Datatype to Larger Datatype
=> which can Call by Implicitly and also Explicitly

Narrowing => used to change Large Datatype to Smaller Datatype
=> which can only Call by Explicitly (Precision loss)

NOTE: Boolean Datatype can't able to do casting.

boolean 1bit | char 16 | byte 8 | short 16 | int 32 | long 64 | float 32 | double 64

```
Public class MainClass1
{
  public static void main(String[] args) {
    System.out.println("---");
```

```
[ byte b = 21;
  int x = (int) b; //Explicit widening
  int x = b; //Implicit widening ]
```

```
[ char ch = 'A';
  int x = (int) ch;
  int x = ch; ]
```

[int i = 22;
// double x = i;] // implicit Narrowing not possible

[double b = 22.909; // explicitly
int x = (int) d; // narrowing]
s.o.pln(x);

Space = 32
0 - 48
* = 42

ASCII Values
A-65 | Z-90 | a-97 | z-122 | 0-32 | ' ' = 48

psvm (String args[]) {

s.o.pln((long)((Math.pow(3,64))/2 -1));

char ch = 'A';
s.o.pln(ch+1); 66
s.o.pln(ch+ch); 130
s.o.pln(ch++); A
s.o.pln(ch); B
s.o.pln(++ch); C

for(char x='A'; x<='Z'; x++) {
s.o.pln(x);
}

int a = 023; // octa number
s.o.pln(" a);

int b = 0x23; // hexa number
s.o.pln(b);

}

}

class MC

```
static void test (byte x)
    {S.o.pln ("I am byte");
    }
static void test(int x){
    S.o.pln ("I am int");
    }
static void test (long x) {
    S.o.pln("I am long");}
static void test (double x){
    S.o.pln("I am double");}
```

```
for( char i='A'; i<='E'; i++)
 {
 for( char j='A'; j<=i; j++) {
     S.o.pln(i);
 }
 S.o.pln();
}
```

o/p

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

```
for ( char i = 'A'; i<='E' ; i++)
   {
   for ( char j = 'A'; j<=i; j++) {
       S. o.pln (j);
   }
   S.o.pln();
}
```

o/p

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

```
char k='A';
for (char i='A'; i<='E'; i++)
    {
     for (char j='A'; j<=i; j++) {
          S.o.p(k);
          k++;
     }
    S.o.pln();
}
```

o/p

A
B C
D E F
G H I J
K L M N O

```
int s=5
int alpha=65
for(int i=0; i<size; i++){
    for(int j=0; j<size-i-1; j++){
        s.o.pln(" ");
    }
  for(int k=0; k<2*i+1; k++){
        s.o.pln((char)(alpha+k));
    }
    s.o.pln();
}}}
```

o/p

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

Type Casting

→ Converting the type of data to any other type of data is known as Type Casting

→ They are two types of Type Casting

i) Widening
i) Primitive Type Casting
ii) class / Reference Casting

Primitive Type Casting

→ Converting one primitive Datatype to any other primitive is known as Primitive Type Casting

→ They are two types of Primitive Type Casting

1.) Widening → Converting Smaller Datatypes to Larger Datatypes

→ Widening Can be done Implicitly by the Compiler (or) Explicitly by the Programmer by using Casting Operator ( ).

2.) Narrowing → Converting Larger Datatypes to Smaller Datatype

→ Narrowing is Must be done always explicitly because it leads to Precision Loss

NOTE :- The Datatype which is not possible to Cast is Boolean

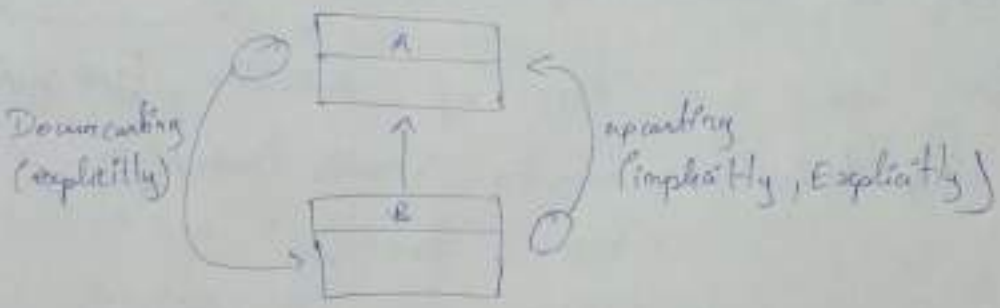
ASCII

A = 65 || Z = 90 || a = 97 || z = 122 || ' ' = 32 || '0' = 48

=> If Datatypes are having Same Size, if the Conversion is adding the precision that is called as widening or else if it is Narrowing

-> The Conversion Class to Short & Short to Char is always explicit

Class / Reference Casting:



Up Casting ex

```
A a1 = (A)new B() //Explicit upcasting
A a1 = new(B) //Implicit upcasting.
B b1 = new();
A a1 = (A) b1; // Explicit upcasting
A a1 = b1; // Implicit upcasting
```

Package Casting

```
class A
{
int x = 10;
}
class B extends A { int y = 20; }.
```

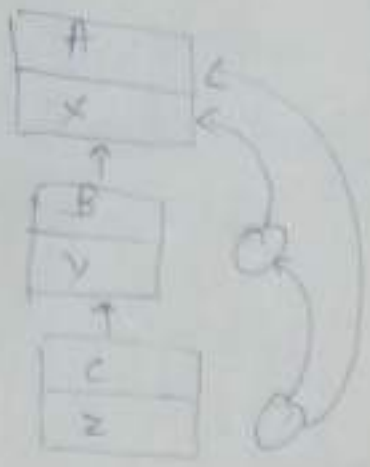
```
class Main {
  psvm(String args[]){

    B b1 = new(B);
    s.o.pln(b1.x);
    s.o.pln(b1.y);

    A a1 = b1; //Implicit upcasting
    s.o.pln(a1.x);

  }
```

```
class c extends B
    {int z = 30}
```

```
class Main1 {
  psvm(String args[]) {

    C c1 = new C();
    s.o.pln(c1.x);
    s.o.pln(c1.y);
    s.o.pln(c1.z);

    B b1 = (B) ~~new C()~~ : c1;
      s.o.pln(c1.x);
      s.o.pln(c1.y);

    A a1 = (A)b1;
      s.o.pln(a1.x);

    A a11 = A new C();
    s.o.pln(a11.x);

  }
```

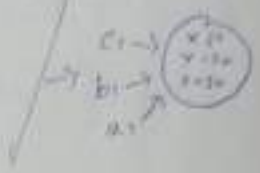
O/P

10
20
30
10
20
10
10

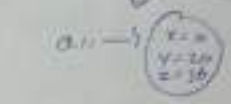


upcasting (implicitly / Explicitly)

```
C c1 = new C();
B b1 = c1;
A a1 = b1;
```



```
A a1 = new C();
```



Downcasting (Restoring object)

only the upcasting object can only be able to perform Downcasting without that it shows classcast Exception error.

```
class me{ psvm(String args[]){
    // downcasting
    // B b1 = (B) new A(); // class cast Exception
    A a1 = new C();
    S.o.pln(a1.x); // upcasting

    B b1 = (B) a1; // downcasting
    s.o.pln(b1.x);
    s.o.pln(b1.y);
C   C1 = (C)b1; // downcasting
    s.o.pln(c1.x);
    s.o.pln(c1.y); s.o.pln(c1.z)
```

Class / Reference Casting

* Converting One object type to another object type or converting One class type to another class type is known as Class / Reference Casting
* To achieve class casting Inheritance is mandatory
* There are two types of class casting
  1) Upcasting
  2) downcasting

1) upcasting

~~-> Converting one object type~~

-> Converting Sub class object into Super class type is known as upcasting

(or)

-> If a Sub class object is Accessed by Super class reference Variable then it is known as upcasting

-> Upcasting Can be done Implicitly by the Compiler or ~~and~~ explicitly by the Programmer by using Casting operator

-> We can achieve upcasting Upto Nth level of Inheritance

-> If the object is upcasted it Shows only Super class properties and behaviors.

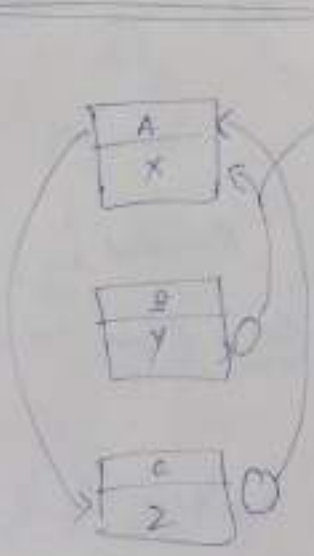
2) Downcasting

-> Converting Superclass object into Sub class type is known as downcasting.

-> Downcasting must be done always Explicitly

(*) -> Only upcasted objects can be downcasted (or) else it shows ClassCastException error

-> The upcasted object shows only Super Class properties and behaviours, if we want to retrace the object we must perform downcasting.



void test (A a1) ← test(new B());
{
C c1 = (C) a1
}
X ClassCastException

A a1 = new A()
A a1 = new B()
A a1 = new C()

void test (A a1)
{
C c1 = (C) a1; ✓
}

test(new A());
test(new B());
test(new C());

Explicit upcasting

So) B and C are upcasting Implicit

```
void test(A a1) {
    if (a1 instanceof C)
    {
      // downcasting
    }
    else if (a1 instanceof B)
    {
      // downcasting
    }
    else { // for object

    }
}
```

```
class Executor {
    void test(A a1){
        if(a1 instanceof C){
            C c1 = (C) a1; // downcasting
            S.o.println(c1.x);
            S.o.println(c1.y);
            S.o.println(c1.z);
        }
        else if (a1 instanceof B) {
            B b1 = (B) a1; // downcasting
            S.o.println(b1.x);
            S.o.println(b1.y); }
        else {
            S.o.println(a1.x);
        }
}
```

```
class MC{
    psvm (String args[]){
        Executor e= new Executor()
        e.etest(new B()); //Implicit upcasting
        e.test(new A());
        e.test(new C()); //Implicit upcasting
    }
}
```

o/p

```
10
20
10
10
20
30
```

NOTE:- If a Method argument is a Super class type we can store Same class object as well as Sub class object, while Storing Passing the Sub class objects those objects are Explicitly upcasted to Super Class type.

* Inside the Methods these objects shows only Super Class Properties and behaviors If you want the Sub class Properties back we must perform down casting.

- Before doing downcasting it is a better practise to check which object Parent in Reference Variable by using [instanceof] keyword.

[to check object Parent in Reference Variable]

Method Overriding with Upcasting

1) Once we Override the Method Even if you perform upcasting we will get Overriden Implementation.

```
class A {
    void wish(){
        S.o.println("Hi");
    }
}
class B extends A{
    void wish(){
        S.o.println("Bye");
    }
}

class MC {
    PSVM(String args[]){

        A a1 = new A();
        a1.wish();

        B b1 = new B();
        b1.wish();

        A a2 = new B(); //upcasting
        a2.wish();
```

o/p
Hi
Bye
Bye

Polymorphism (बहुरूपता)

- Compile Time
  - Early Binding
  - Static Binding
- Run Time
  - Late Binding
  - Dynamic Binding

Polymorphism -> An object showing Different behaviours in its life Cycle

Polymorphism types -> Compile Time Polymorphism
Run Time Polymorphism

Compile Time Polymorphism

-> The Method Declaration will Binds with Method Definition by the Compiler at Compile time, this is known as Compile time Polymorphism

=> Compile Time Polymorphism is also known as Early Binding or Static Binding

=> Before the Execution the method Declaration is binded with its definition hence it is called as Early Binding

=> Once the declaration is Binded with the definition it can't be rebinded. Hence it is called as Static

Method Overloading

Run Time Polymorphism

=> In Run time Polymorphism the Method Declaration will be binded with Method definition by the JVM at Run time.

=> To achieve Run Time Polymorphism the following things are mandatory

1.) Inheritance
2.) Method Overriding
3.) Upcasting

=> The Run time Polymorphism is also known as Late Binding or Dynamic Binding

↓

At the time of execution the Method Declaration will be binded with Method Definition Hence it is called as Late Binding

↓

Once the Declaration is binded with Definition it Can be rebinded. Hence it is Called as Dynamic Binding

method Declaration

→ modifier return-type MethodName (arguments)

method Definition → { ≡ return value; }

Method

Concrete method: void m1() { ≡ }

abstract method: abstract void m1();

Class

Concrete class: class A { }

abstract class: abstract class A { }

eg:- abstract class A {

abstract void m1();

abstract void m2();

void m3()

{

sop("hi");

}

}

class B extends A {

void m1() {

≡

}

void m2() {

≡

}

}

class MC { PSVM(S a[]) {

B a1 = new A; a1.m1()

A a1 = new B; a1.m2()

a1.m3()

```
abstract class Calculator
{
abstract void add(int a, int b);
abstract void Sub(int a, int b);
}

class MyCalculator extends Calculator {

    void add(int n1, int n2){
        S.o.println(n1+n2)
    }
    void sub(int n1, int n2){
        S.o.println(n1-n2)
    }
}

class MC{
    PSVM(String args[]){

        Calculator c1=new MyCalculator();
        c1.add(10,20);
        c2.Sub(100,50);

    }
}
```

o/p

30
50

Method

∞ If Method Contains Both Declaration and Definition then it is Concrete method.

∞ If Method Contains Only Declaration then it is Abstract method.

⇔ Abstract Method must be defined with Abstract keyword

class

⇒ If a class Contains Only Concrete methods then it is Concrete class.

→ Any class if it is defined with Abstract keyword then it is Abstract class.

→ Inside the Abstract class we can define Only Concrete methods (or) Only Abstract methods (or) Both methods.

→ It is never possible to Create a object for Abstract class. [This is the Only difference b/w Concrete class and Abstract class.

→ In a class if we define Any Abstract method, then we must define that class as Abstract

-> Whenever we want to Override the Method in the Sub class we must Define that class as ---> Method as Abstract.

-> We Can Inherit from an Abstract class by using Extends keyword

-> While Inheriting from the Abstract class we must Override all the Abstract Methods or else we should define the Sub class as abstract

-> We Can Create the object of Sub class, we can Store it in a Abstract class Reference Variable and we Can Use (upcasting).

-> The following keyword Combinations are not Possible

i) Private / Abstract
ii) Static / Abstract
iii) Final / Abstract

-> Inside an Abstract class we can also have Concrete methods hence it is not Called as 100% Abstraction.

Interface

Syntax

Interface Interfacename {

-> default abstract method.

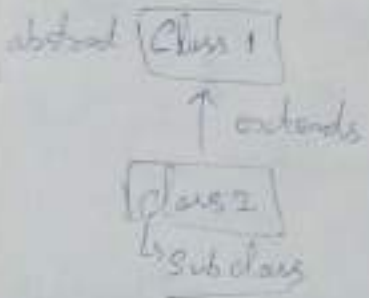
-> Data members are defaultly static & final

-> No Constructor, block, Non Concrete method.

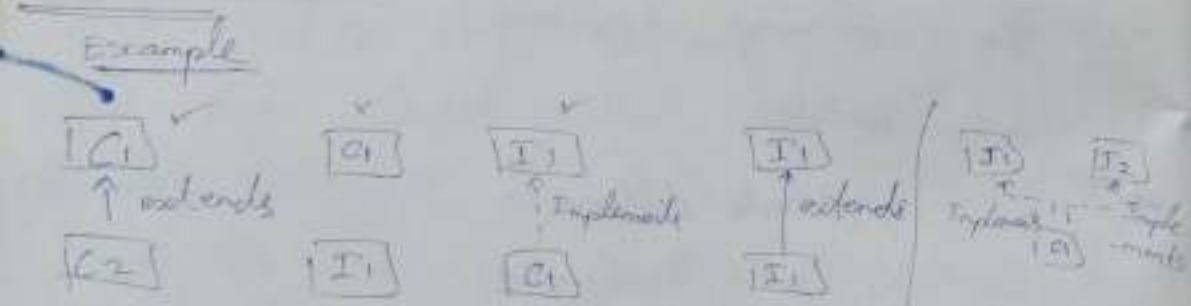
}

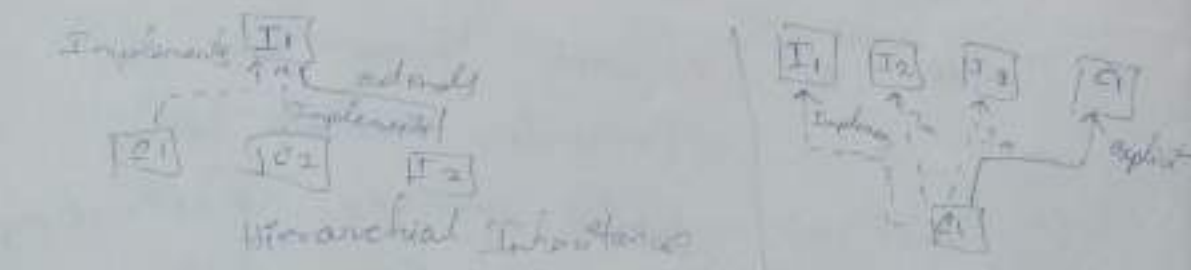
DM → static & final
MM → abstract & Public

Diff b/w Interface and class?



Example





-> Multiple Inheritance is Supported in Interface

```
Interface A {
    int a=10;
    void Add();
    void Sub();
}
Interface B {
    void mul();
}
class c implements A, B {
  Public void Add() {
        System.out.println("ABC");
    }
  Public void Sub() {
        System.out.println("XYZ");
    }
  Public void mul() {
        System.out.println("FOR");
    }
}
class mainclass {
    Psum(String args[]) {
        A a1 = new C();
        a1.Add();
        a1.Sub();
        B a2 = new C();
        a2.mul();
    } System.out.println(A.a);
}
```

o/p

ABC
XYZ
FOR
10

After Java 1.7 version

Concrete ~~only~~ static and Default method can be used in Interface.

methods like ↗

```
Interface A {
    void m1();
Static void m2() { S.O.pln("A"); }
default void m3() { S.O.pln("B"); }
}
```

~~interface B es.~~

```
class B Implements A {
    public void m1() { S.O.pln("K"); }
}
class MC { PSVM (String args[]) {
    A a1 = new B
        a1.m1();
        a2.m2();   A.m2();
        a3.m3();   a1.m3();
    }
}
```

o/p

K
B
A

Class classname{

DM
+
FM
+
blocks
+
Constructors

}

interface interfacename{

-> Data Member are static & final.

-> Funtion Prototypes are abstract

-> no blocks / Constructors / Concrete methods

-> default Access is public

}

[Interface is a definition block to achieve Abstraction]

eg interface A

{

int x = 10;

void m1();

void m2();

}

class B implements A {

{

public void m1(){

=

}

public void m2(){

=

}

}

class Main{ PSVM(String args[]){

System.out.println(A.x);

A a1 = new A(); (X)

A a1 = new B();

a1.m1();

a2.m2();

}

}

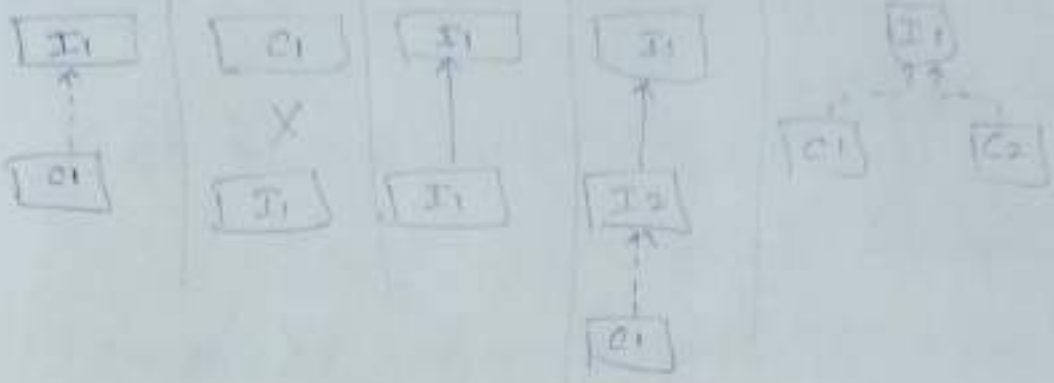
```
Interface Calculator {
    double PI = 3.14;
    double m1(double n1, double n2);
    double m2(double n1, double n2);
}
class MyCalculator implements Calculator {
    double m1(double n1, double n2) {
        return n1+n2;
    }
    double m2(double n1, double n2) {
        return n1 - n2;
    }
}
class MainClass { PSVM (String args[]) {
    Calculator . PI;
    Calculator C1 = new MyCalculator();
    S.O.println(C1.m1(10, 20))
    S.O.println(C2.m2(30, 10));
    }
}
```

o/p

3.14

30.0

20.0



Interface

=> Interface is a Definition block which is used to achieve Abstraction

=> Inside the Interface by default

DM are -> Static and final (Global Constants)

FM are -> abstract

We can't define block or Constructors or Concrete method in the Interface and the default access is Public

=> We can access DM of an Interface by Static reference Syntax, that is

Interface . data member

=> It is Never Possible to Create the object for Interface.

=> We Can Inherit from an Interface by Implements keyword

=> while Inheriting from an Interface we must override all the abstract methods or else we Should define the implemented class as Abstract

=> We Can Create the object for Implementation class and we Can access Through Interface reference Variable (Upcasting)

=> We Can Inherit from Interface to Interface by extends keyword

=> It is never Possible to Inherit from Class to Interface

=> * [Java Support Multiple Inheritance through Interfaces] *

=> A class can extends from only one class but it can Implement from multiple Interfaces

- extends ---- Implementation

Advantages

1) It is used to achieve Abstraction
2) It is used to achieve multiple Inheritance
3) Used to achieve Generalization

Abstraction

-> Hiding the implementation details and showing the necessary Behaviours is known as Abstraction

=> We can achieve Abstraction by Abstract class & Interfaces

-> To achieve 100% Abstraction we use Interface

-> Steps to achieve Abstraction

i) Generalize the Sub class Behaviours in an Interface

ii) Provide the implementation in the Implementation class

iii) Create the Implementation class object and access through Interface reference Variable

* Because of abstraction loose coupling of an application * loose Coupling is a way of Programming If we do the changes in one layer of application it will not have much Impact on other layer of implementations

WAP to count the Armstrong numbers 100 to 100000

```
{
for (int i= 100 ; i<=100000 ; i++) {
    int count=0;
    int n= i, dc=0 , temp=n , sum=0 , a=n;
    while (a>0){
        dc++;
        a= a/10;
    }
while (n>0){
    int r = n %10;
    sum= sum + Power(r, dc);
    n=n /10;
    }
if (sum == temp)
        count++; }
        S.O.println (count); }
static int Power (int r , int dc){
    int P=1;
    for (int i=1 ; i<=dc ; i++){
        P= P*r;
    }
    return P;
}
}
```

class Palindrome{
Public static void main{
n=121, sum=0, temp=n;
rev

while(n>0){
int r=n%10;
rev=rev*10+r;
n=n/10;
}
if(rev==temp)
S.O.pln("Palindroome");
else
s.o.pln("Not a Palindroome");

- for(int i=10 ; i<=100; i++){
int n=i, sum=0, temp=n;
while(n>0){
int r=n%10;
Sum=Sum+fact(r);
n=n/10;
}
if(Sum==temp)
S.O.pln("It is a strong");
else
S.O.pln("It is not a strong");

3

---

static int fact(int n){
fact1=1;
for(int i=1; i<=n; i++)
fact1=fact1*i;
}return fact1;

```
class Palindrome {
    psvm(String args[]){
        for(int i=1; i<=1000; i++){
            int n=i, rev=0, temp=n, count=0;
            while(n>0)
            {
                int rem=x;
                rev = rev*10 + n%10;
                n = n/10;
            }
        if (rev==temp)    // count++;
            {
            s.o.pln(n+" is Palindrome");
            }
        else
            { s.o.pln(n+" not a Palindrome");
        }
        s.o.pln(Count);
    }
}
```

Diff b/w Abstract class and Interface

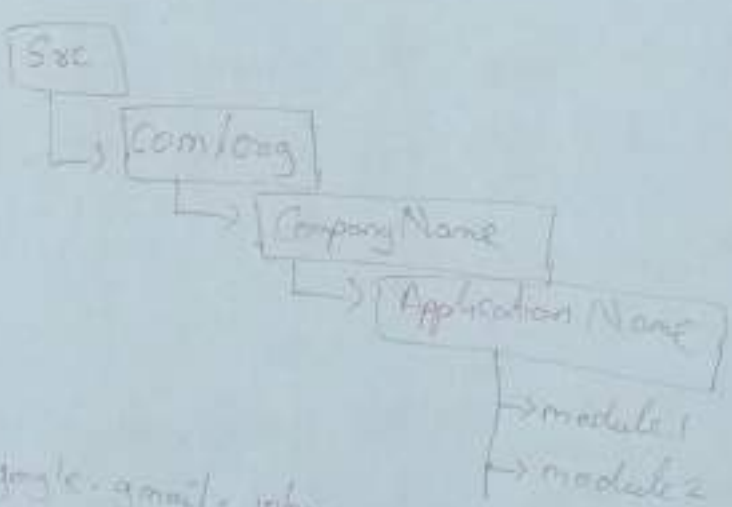
Abstract class	Interface
=> Any class if it defined with Abstract keyword then it is Abstract class.	=> Interface is a Definition block which used to achieve Abstraction
=> Abstract class does not Support multiple Inheritance	=> Interface Support multiple Inheritance
=> Abstract class contains Data Member and Constructors	=> Interface does not ~~Support~~ Contains Data Members and Constructors
=> An abstract class Contains both incomplete (Abstract) and Complete member	=> An interface Contains only incomplete member (Signature of member)
=> An abstract class can Contain access modifiers for the Subs, functions, Properties	=> An Interface Cannot have access modifiers by default everything is assumed as public
=> Only Concrete Member of abstract class can be Static	~~Memory of~~ . => Member of Interface Can not be Static

Interface

-> Normal Interface
-> Functional Interface (only one abstract method)
-> Marker Interface

Java Packages

It is a folder which is usually containing Java and Java Related files



Eg:-
com.google.gmail.inbox;

=> Every Java file first line must be package declaration

=> Package name should be start with small letters

```
package com.google.gmail.inbox;
Public class Message {

}
```

1 => We can access classes directly if the class are belongs to same Package.

eg:-

```
package com.google.gmail.inbox;
public class Message
{
    Spam ();
}
```

I => If we want to use the classes which is present in another package we must write fully qualified Classname

(package name

eg:-

```
package com.google.gmail.login;
public class Login {
    com.google.gmail.inbox.Spam s = new com.google.gmail.inbox.Spam();
}
```

=> If we want to access the classes which is present in another Package we should write fully qualified import statement

eg:-

```
package com.google.gmail.login;
import com.google.gmail.inbox.Spam;
{
}
```

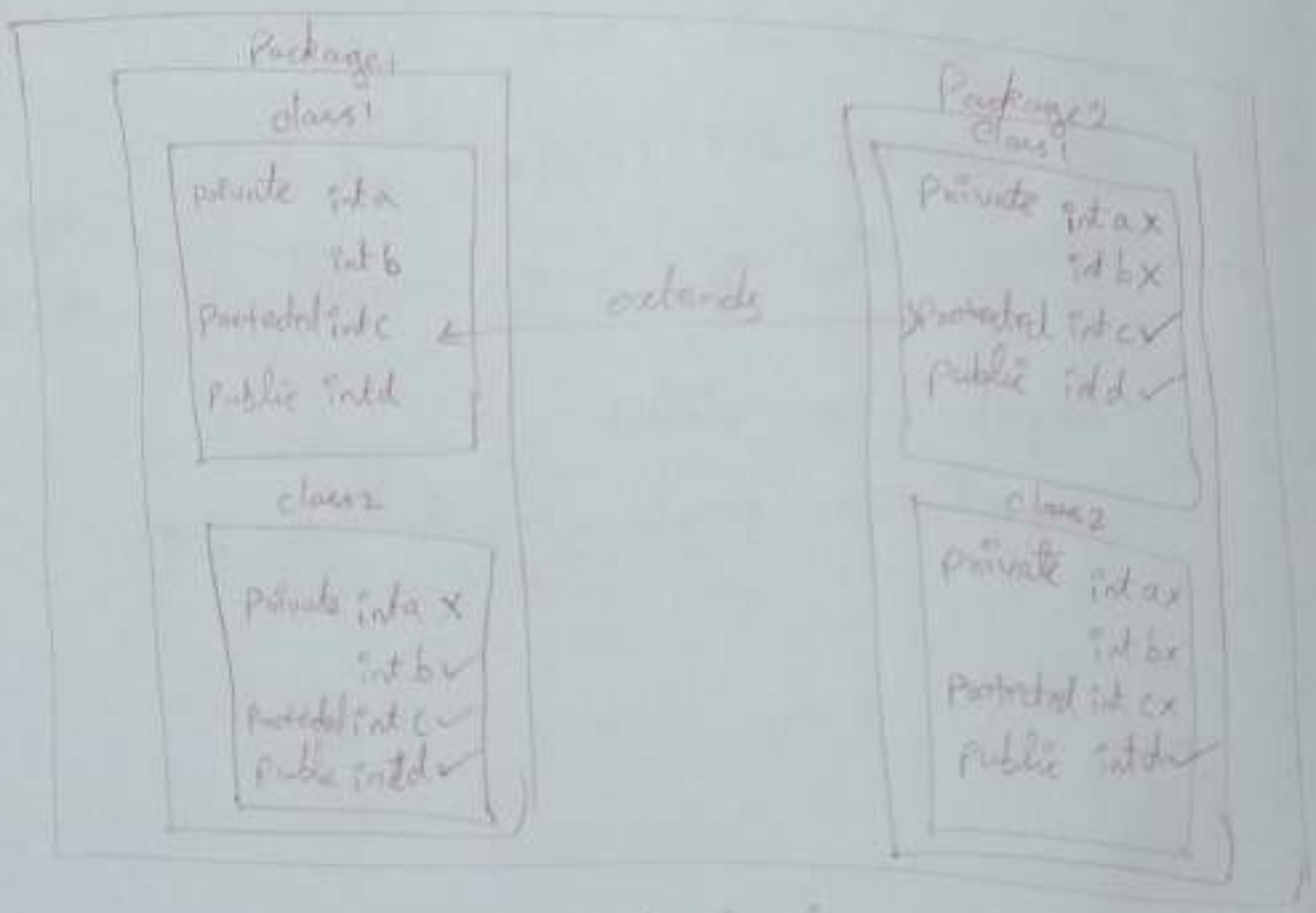
c) Import Statement must be written outside the class and inside the Package Definition

```
package com.google.gmail.Login;
import com.google.gmail.Index.*;
Public class Login
    {
    Spam S1;
    Spam S2;
    Message m1;
    }
```

d) If we want to Import all the classes from the Package we must use Astrick

```
import com.define.
Import com.google.gmail.Index.*;
```

ENCAPSULATION



```
Package pack1;
public class Sample1 {
    private int a = 10;
    int b = 20;
    protected int c = 30;
    public int d = 40;

    PSVM(String[] args) {
        Sample1 s1 = new Sample1();
        s1.println(s1.a);
        s1.println(s1.b);
        s1.println(s1.c);
        s1.println(s1.d);
    }
}
```

```
Package pack1;
public class Sample2 {
    PSVM(String[] args) {
        Sample1 s1 = new Sample1();
        //s1.println(s1.a); not possible (Private)
        s1.println(s1.b);
        s1.println(s1.c);
        s1.println(s1.d);
    }
}
```

```
Package pack2;
import pack1.Sample1;
public class Sample3 extends Sample1 {
    PSVM(String[] args) {
        Sample3 s1 = new Sample3();
        //s1.println(s1.a);
        //s1.println(s1.b);
        s1.println(s1.c);
        s1.println(s1.d);
    }
}
```

```
Package pack2;
import pack1.Sample1;
public class Sample4 {
    PSVM(String[] args) {
        Sample1 s1 = new Sample1();
        //s1.println(s1.a);
        //s1.println(s1.b);
        //s1.println(s1.c);
        s1.println(s1.d);
    }
}
```

Compiler

- Source code to byte code
- Void Return type
- JVM tank (garbage collector)
- Inheritance (object class)
- Super() for zero param Constructor and Parameterized Constructor. ~~other~~ of Super class Constructor.
-

ABSTRACTION

-> Hiding the implementation details and showing the necessary Behaviours is known as Abstraction

-> We can achieve Abstraction by Abstract class of Interface

=> To achieve 100%. Abstraction we use interface

-> Steps to achieve Abstraction

1 -> Generalize the Sub class Behaviours in an Interface

2 => Provide the Implementations in the Implementation class

3 => Create the Implementation or class object and access throw Interface reference Variable

[Because of abstraction Loose Coupling of an application] Loose Coupling is a way of Programming if we do the changes in one Layer of Application it will not have much Impact on other Layer of Implimentation.

Diff blw Abstract class and Interface

Abstract class

=> Any class if it defined with Abstract keyword then it is Abstract class.

=> Abstract class does not Support multiple inheritance

=> Abstract class Contains Data Member and Constructors

=> An abstract class Contains both incomplete (abstract) and Complete member

=> An abstract class can Contain access modifiers for the Subs, functions, Properties

=> Only Concrete Member of abstract class can be Static

Interface

=> Interface is a Definition block which used to achieve Abstraction

=> Interface Support multiple Inheritance

=> Interface does not Support Contains Data Members and Constructors

=> An interface Contains only incomplete member (Signature of member)

=> An Interface Cannot have access modifiers by default everything is assumed as public

~~Members of~~

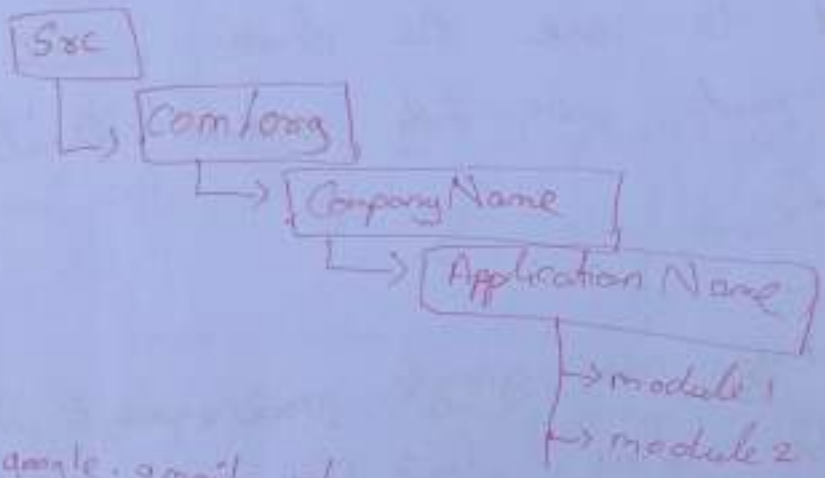
=> Member of interface can not be Static

Interface

→ Normal Interface
→ Functional Interface (only one abstract method)
→ Marker Interface

Java Packages

It is a folder which is usually containing Java and Java Related files



Eg:-
com.google.gmail.inbox;

=> Every Java file First line Must be Package declaration
=> Package name should be start with Small letters

```
package com.google.gmail.inbox;
public class Message {

}
```

=> We can access classes directly if the classes are belongs to same package.

eg:

```
Package com.google.gmail.inbox;
public class Masag
{
Spam s1;
}
```

=> If we want to use the classes outside the package we must write fully Qualified package name.

eg:-

```
package com.gorgle.gmail.login;
public class login {
    com.google.gmail.inbox.Spam s1;
}
```

=> If we want to access the classes outside the Package we should write fully Qualified import Satatement

eg:-

```
package com.google.gmail.login;
import com.google.gmail.inbox.Spam;
{
}
```

-> Import Statement must be written outside the class and inside the package Declaration

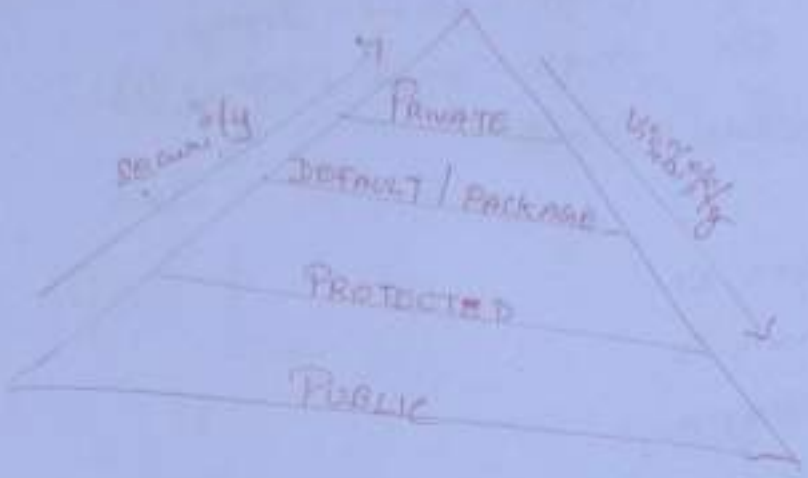
```
package com.google.gmail.login;
import com.google.gmail.index.*;
Public class login
    {
    Spam s1;
    Spam s2;
    Masage m1;
    }
```

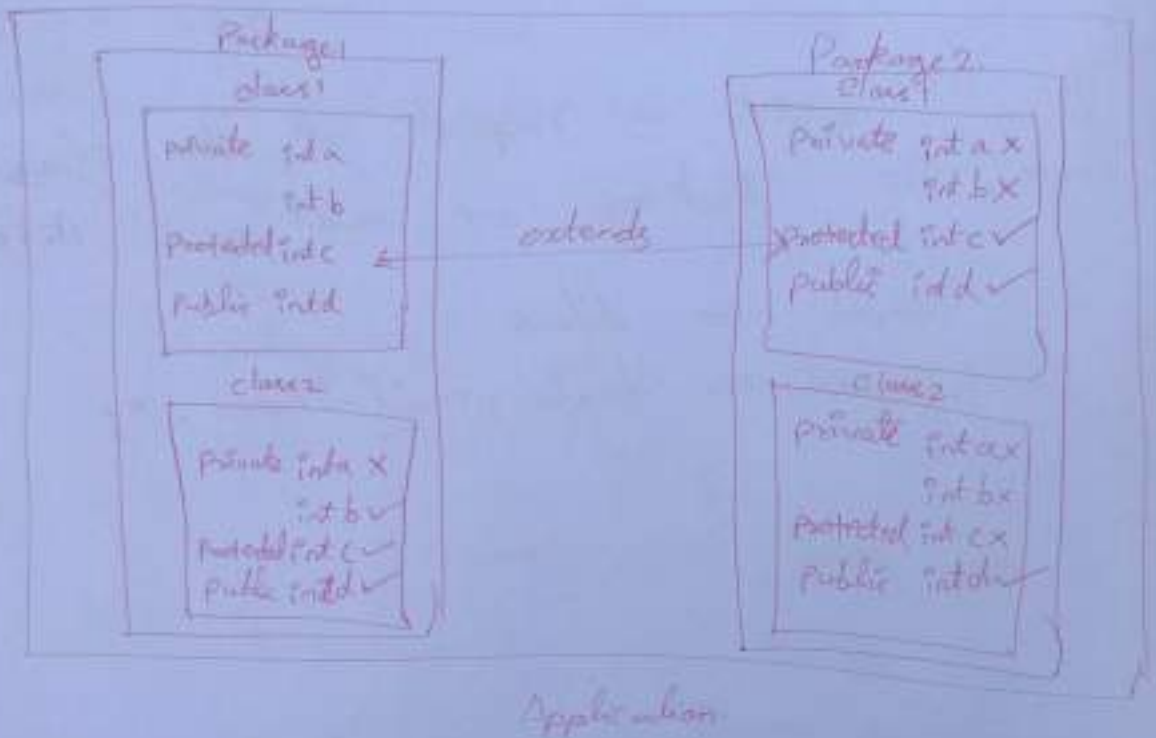
-> If we want to Import all the classes from the package we must use Astrick

~~import com.disno~~

```
import com.google.gmail.index.*;
```

ENCAPSULATION





Application

```
Package pack1;
public class Sample1{
    private int a=10;
    int b=20;
    protected int c=30;
    public int d=40;
PSVM(String[] args){
    Sample1 s1 = new Sample1();
    s.o.pln(s1.a);
    s.o.pln(s1.b);
    s.o.pln(s1.c);
    s.o.pln(s1.d);  } }
```

```
Package pack1;
public class Sample2{
    PSVM(String[] args){
        Sample1 s1 = new Sample1();
        //s.o.pln(s1.a); not possible (Private)
        s.o.pln(s1.b);
        s.o.pln(s1.c);
        s.o.pln(s1.d);
    }
}
```

```
Package pack2;
import pack1.Sample1;
public class Sample3 extends Sample1{
    PSVM(String args[]){
        Sample3 s1 = new Sample3();
        //s.o.pln(s1.a);
        //s.o.pln(s1.b);
        s.o.pln(s1.c);
        s.o.pln(s1.d);
    }
}
```

```
Package Pack2;
import pack1.Sample1;
public class Sample4{
    PSVM(String args[]){
        Sample1 s1 = new Sample1();
        //s.o.pln(s1.a);
        //s.o.pln(s1.b);
        //s.o.pln(s1.c);
        s.o.pln(s1.d);
    }
}
```

Wrapping the Data inside Some Definition block is known as encapsulation

(or)

Protecting the member of the class by using Some access specifier

→ We have 4 access specifiers

Private, Default, Protected, Public

Private => * The private members can be accessed only within a class

* It is highly secured access and Less visible access

Default / Package => If we are not mention any access specifier then the access will be Default or Package.

=> This member can be accessed only within class and within a Package

=> Not

Protected => The Protected members can be accessed within a class, within a Package and also outside the package by inheritance

public => Public member can be accessed by anywhere

=> It is less secure and high visible access

Java Jar

Java Application must be convert as Jar files

Jar stands for (Java archive)

It is a compressed folder which usually contains .class files & and resources

How to create a Jar file (export)

-> Right click on project / Go export / Select java / Select Jar / and browse the jar file destination / click on finish.

How to add jar file into java build path (import)

-> Right click on project / Go to properties / Select java build path / Select libraries / Select class Path / click Add to external jar / Browse the jar file / Apply / close.

-> click on jar file run as java administrator

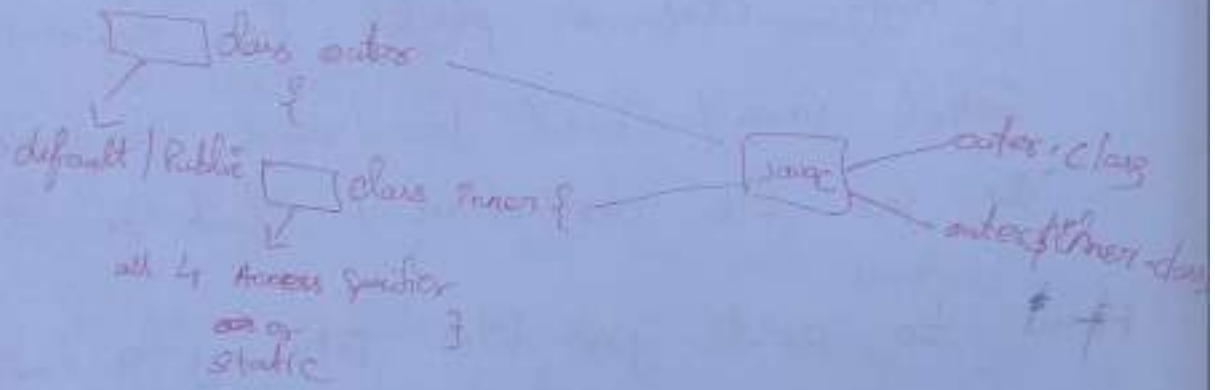
WAR => Web archieve
EAR => Enterprise archive

Rules of Encapsulation [Rule1]

1.) We Can Define a class inside the class that is known as Nested class or Inner class

2.) The outer class can have only 2 access specifiers
   1.) Default / Package
   2.) public

3.) Inner class can have all the 4 access specifiers including static



Rule 2

1.) In a file we can define 'N' number of class and the file name must be same as public class name.

2.) It is never possible to define more than one public class in file

```
class outer
{
  class Inner
  {
    int a=10;
  }
PSVM(S arg[])
  {
    outer.Inner ril=new outer().new Inner();
    S.O.Pln(ril.a);
  }
}
```

```
class A
{

}
class B
{

}
Public class C
{PSVM(S arg[])
}
```

i) If we have main method not can specify Public access specifier to that class

ii) Public should only be defined in own class file

Rule 3

i) Default Constructor Access is always same as class access. If we create our own Constructor we can give all the 4 Access Specifiers

Note:- i) If I define the Constructor as private we can't create a object outside the class

ii) If the Constructor private we can't achieve inheritance (Constructor chaining not possible)

private Constructors
→ can't create object
→ Constructor chaining not possible

=>

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

4 Access specifiers

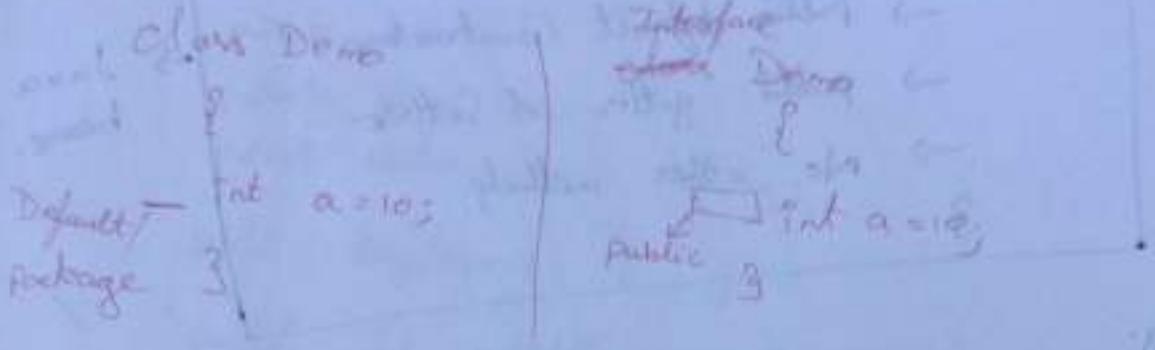
```
Public class A
{
    A() {
    }
}
```

for default Constructor it will take same access specifiers for it

if we are creating Constructors all 4 access specifiers can be used by programmer

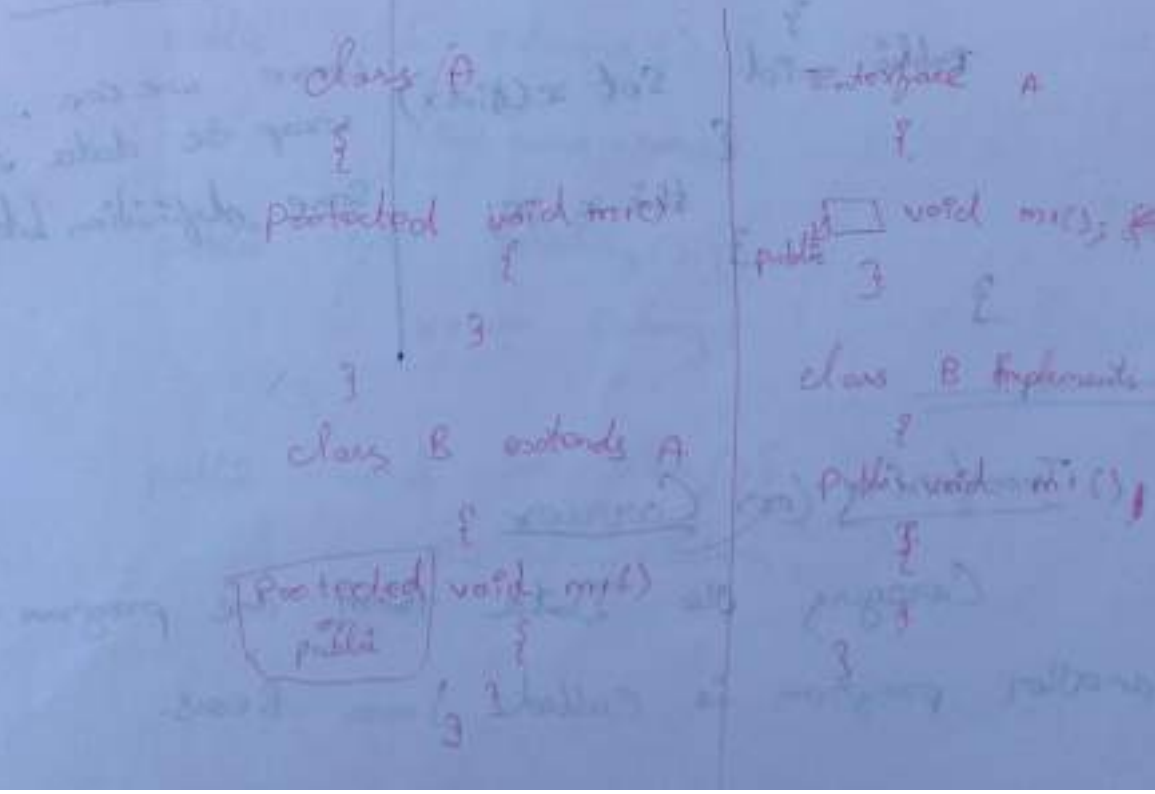
Rule 4

1.) The Default Access inside the classes Default or Package and Default access inside the Interface Public



Rule 5

1.) While performing the method overriding as a programmer we can maintain the visibility (or) we can increase the visibility but we can't reduce (or) the visibility



DTO
Data Transfer object (or) JAVA BEANS (or) POJO
Playing with old java objects

```
class classname
{
  -> Private Data member
  -> public default Constructor
  -> public getter & Setter
  -> No other methods
}
```

-> Java beans.

eg:

```
public class B
{
    private int x = 10;
    Public A() { }
    public int getx()
    {
        return x;
    }
    public void Setx(int x)
    {
        this.x = x;
    }
}
```

=> Encapsulation example.

where we can wrap the data inside some definition block

Transfering (or) Carrier

Carrying the Data from one program to another program is called Java Beans.

Same package or diff pkg

```
package encapsulation;
class public class Employee
{
    private int id;
    private String name;
    private ~~Salary~~ double Salary;
    Public Employee() { }
    public int getId(){
        return id;
    }
    public void setId(){
        this.id = id;
    }
    Public int getname() {
        ~~this~~ return name;
    }
    public void setname() {
        this.name = name;
    public int getSalary() {
        return Salary;
    }
    public void Set_Salary(){
        this.Salary = Salary;
    }
```

```
class mc {
    psvm (S args[])
    {
        Employee emp = new Employee();
            emp.setId(153);
            emp.setname("Bhavi");
            emp.setSalary(45000.0);
        S.o.pln(getId emp.getId());
        S.o.pln(emp.getId getname());
        s.o.pln(emp.getSalary());
    }
}
```

o/p

153
Bhavi
45000.0

Java beans (POJO) (DTO)

-> It is a class with Private Data Members, public default Constructors, public getters & Setters and no other methods other the getters and setters

=> Java beans is a very good example encapsulation.

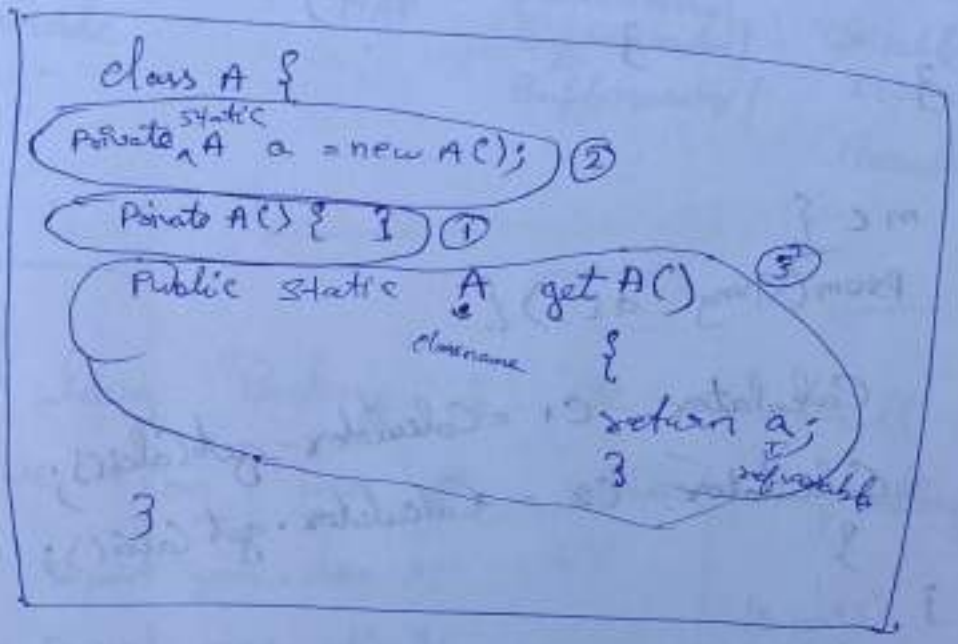
=> It is used to Transfer the data from one program to another program Hence they called as Data Transfer objects (DTO's)

Singleton

=> Singleton is one of the Design Patterns which will have or allows to Create Only One Object of class

=> We Can acheive Singleton by following Steps

1.) Define the Constructor as Private
2.) Create a Single object within a class
3.) Create a Static method which returns Same reference.



Design Pattern => It Contains proven Solution

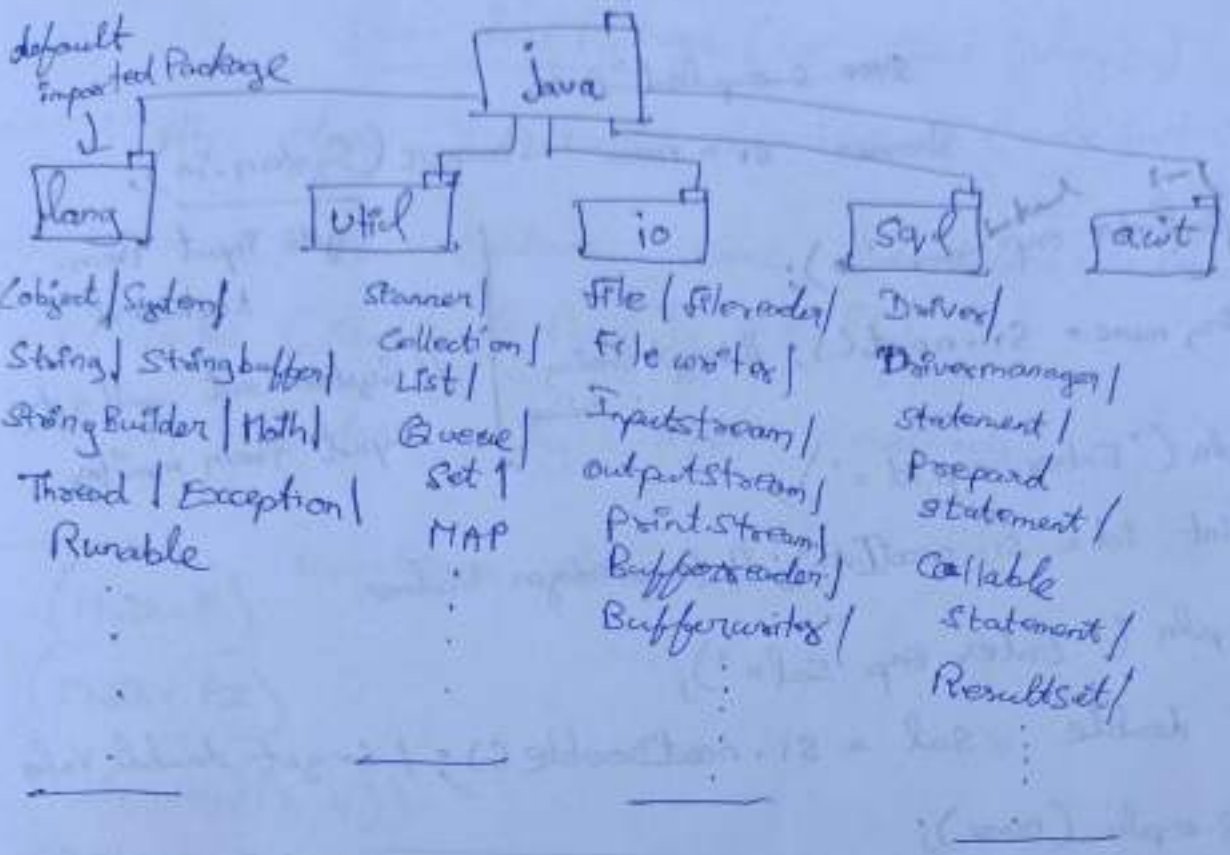
```
Package encapsulation{
public class Calculator
    {
    private static Calculator calci = new Calculator();
    Private Calculator()
        {
        s.o.p("Calculator object is Created..!!");
        }
    Public Static Calculator getCalci()
            {
            return calci;
            }
}

class mc {
    Psvm(String a[]){
     Calculator c1 = Calculator.getCalci();
     Calculator c2 = Calculator.getCalci();
     }
}
```

O/P

Calculator object is Created..!!

Core Java [(lang, util, io) Packages]

Java Libraries



=> lang Package is the default One.
we can use it without importing it

```
import java.lang.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import java.awt.*;
```

(imports) * includes every classes present inside the Packages.

```
import java.util.Scanner;
public class Employee {
    Psvm(String args[]) {
        ~~Syso S.o.pln("Ente~~
        Scanner S1 = new Scanner(System.in);
S.o.pln("Enter emp Name=");
String name = S1.next(); // to get String value
S.o.pln("Enter emp Id =");
    int id = S1.nextInt(); // to get Integer Value.
S.o.pln("Enter emp Sal=");
    double Sal = S1.nextDouble(); // to get double Value
    S.o.pln(name);
    S.o.pln(id);
    S.o.pln(Sal);
    }
}
```

gets input from keyboard

// System.out will get input from monitor

o/p

```
Enter emp Name = Bhavi
Enter emp id = 567
Enter emp Sal = 50000.0

Bhavi
567
50000.0
```

=> nextLine() method will read Complete line what we entering //

Utilities

⇒ Math is a utility class which helps us to perform Mathematical Operations

-> Math is a final class present in java.lang.Package (we can't inherit)

-> Math class Constructor is Private (we can't create object)

=> Math class contains some data members (Global Constants [Static & final] (E & PI)

-> It provides only static methods.

---

import java.util.Random;

(Math.E)

(Math.PI)

(Math.Pow(3, 4))

(Math.sqrt(25));

(Math.cbrt(27))

(Math.max(10, 20)

Math.min(4, 1);

Math.floor(4.5)

Math.ceil(4.5)

Math.round(4.5)

Math.abs(-5)

Math.log10(3.5)

Math.random()

```
for(int i=1; i<=100; i++)
    s.o.println(generateOTP());
```

```
Random r1 = new Random();
    S.o.p(r1.nextInt());
}

static String generateOTP() {
    String OTP = " ";
    OTP = OTP + (int)(Math.random() * 10)
              + (int)(Math.random() * 10)
              + (int)(Math.random() * 10)
              + (int)(Math.random() * 10);
    return OTP;
}
```

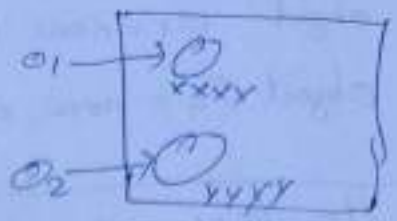
---

object class

→ It is a root class present in java.lang Package
→ ✗ No Data members
→ ✓ Public default Constructors
→ ✓ Non Static Methods are

- toString()
- hashcode() } **
- equals() }
- clone()
- finalize()
- Threads
  - method overloading
    - wait()
    - wait(long ms)
    - wait(long ms, int ns)
  - notify()
  - notifyAll()
  - getclass()

object o1 = new object();
object o2 = new object();



toString()

String s1 = o1.toString();
S.O.pln(s1); // java.lang.object@xxxx

String Representation of object
: [fully Qualified class Name @ address]
↓
Hexadecimal

String s2 = o2.toString()
S.O.pln(s2); // java.lang.object@yyyy

---

hashcode()

int n1 = o1.hashcode();
S.O.pln(n1); // 123456 → [Unique Integer Numb
int n2 = o2.hashcode
S.O.pln(n2); // 57869.5 [JIN]

---

equals()

x boolean res = o1.equal(o2); x
S.O.pln(res); // false
only can Compare with Same class.
✓ boolean res = o2.equals(o2) ✓

```
object o1 = new object();
object o2 = new object();

//explicit Call to toString()
String s1 = o1.toString();
s.o.pln(s1);
s.o.pln(o2.toString());

// Implicit Call toString()
s.o.pln(o1);
s.o.pln(new object());
```

Explicit

toString();

Implicit

```
int n1 = o1.hashcode();
s.o.pln(n1);
s.o.pln(o2.hashcode());
```

hashcode();

```
boolean b1 = o1.equals(o2);
s.o.pln(b1); // false
boolean b2 = o2.equals(o2)
s.o.pln(b2); // True
s.o.pln(o2.equals(o1)); // false
s.o.pln(o1.equals(o1)); // True
```

equals()

```
public class Student implements Cloneable
{
    int id;
    String name;
    double perc;
    public Student (int id, String name, double perc)
    {
        this.id = id;
        this.name = name;
        this.perc = perc;
    }
public String toString()
    {
    return "[id, name, perc]";
    }
Public int hashCode()
    {
        return id;
    }
public boolean equals(Object o1)
    {
    if (this.hashcode() == o1.hashcode())
        {return true;
        }
        else { return false; }
    }
```

```
package objet Class Demo

import com.qsp.stdApp.Student;
public class MC {
    psvm(String[] args)
    {
        Student std1 = new Student(1, 'AB', 87.0);
        Student std2 = new Student(5, 'ZX', 84.0);
        Student std3 = new Student(1, 'AB', 87.0);

    //explicit call toString()
        String s1 = std1.toString();
        S.o.pln(s1)
        S.o.pln(std2.toString());
        S.o.pln(Std3.toString());
    // Implicit Call to String()
        String S.o.pln(std1);
        S.o.pln(new student('3, 'PR', 58.0);

    int h1 = std1.hashcode();
        S.o.pln(h1);
        S.o.pln(std2.hashcode());

    boolean res = std1.equals(std2);
        S.o.pln(res);
        S.o.pln(std1.equals(std2);
```

toString()

-> It returns String Representation of object (fully Qualified Classname@address)

-> toString can be called Implicitly as well as explicitly

-> It will be called implicitly when we try to print ref.var (or) when we try to print object

Syntax:
```
Public String toString()
```

hashCode()

-> It returns hashCode of object

-> Hashcode is a UIN (unique integer Number which is calculated based on address of an object

->
Syntax:
```
public int hashcode()
```

equals()

-> It Compare the objects Based on address, if addresses are equal it returns True or else it returns false.

Syntax:
```
public boolean equals(object o)
```

Note:- We can call these methods by creating an object of object class or by creating the object of any class because every class in Java is a Subclass of object class

→ These Methods will be inherited to all the classes

→ By default toString(), hashCode(), equals() will be working based on address.

→ We can override toString(), hashcode(), and equals(obj) in the Subclass

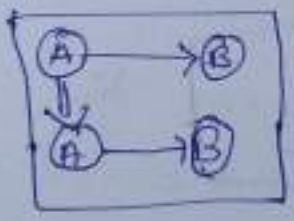
---

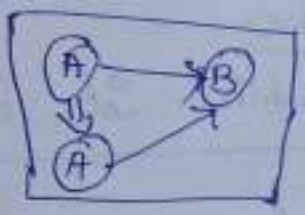
Cloning

→ taking an Exact Copy of Existing object

→ Deep cloning

Shallow cloning (✓)





Interface with Empty body

Cloneable (I) → It is a MarkerInterface Present in Java.lang.

It pass Runtime info to JVM

Student

Employee

X

Account

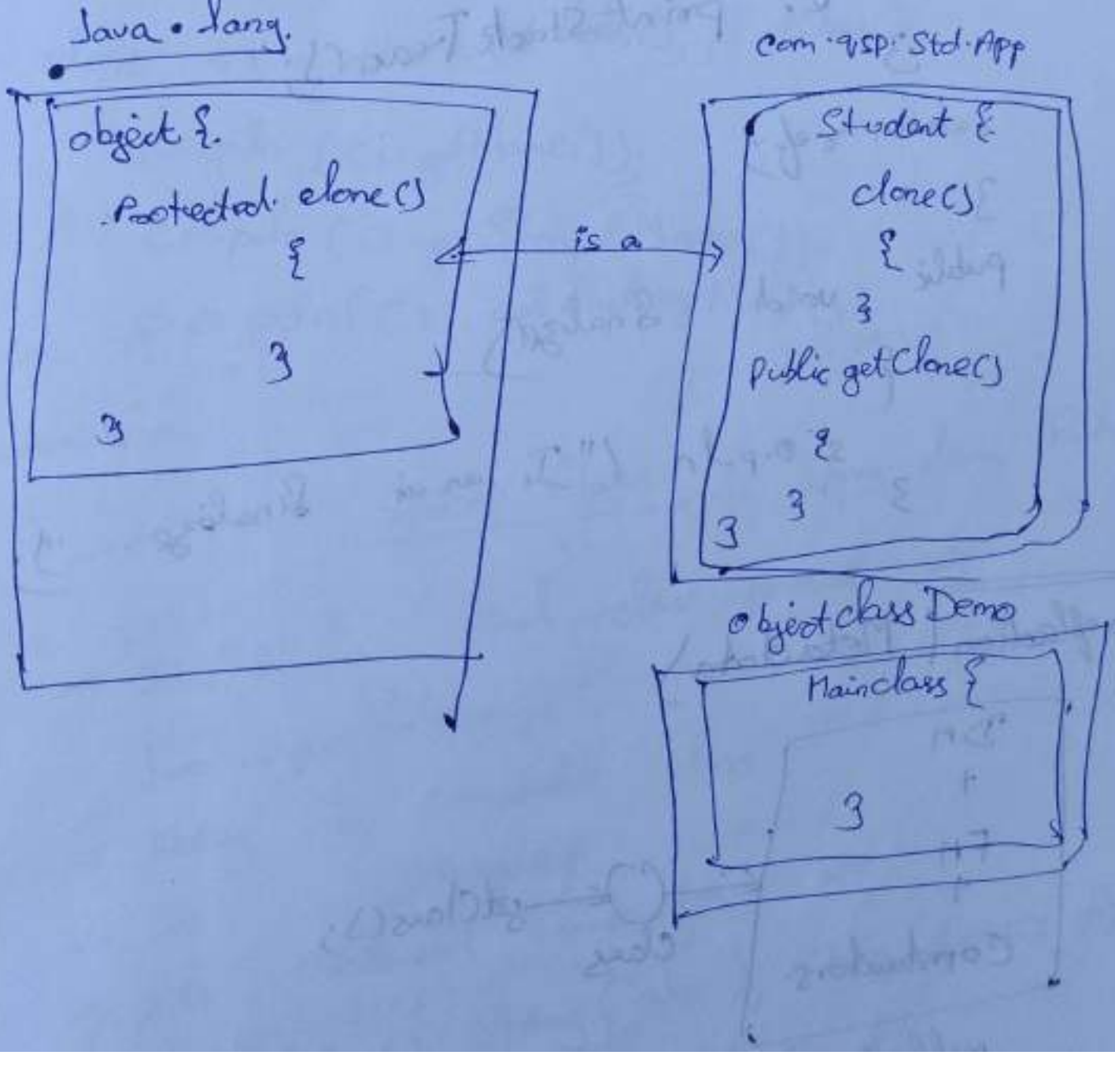
clone() Not Support Exception

clone()

It returns clone of an object which is of type Clonable or else it throws "CloneNotSupported Exception"

Syntax -

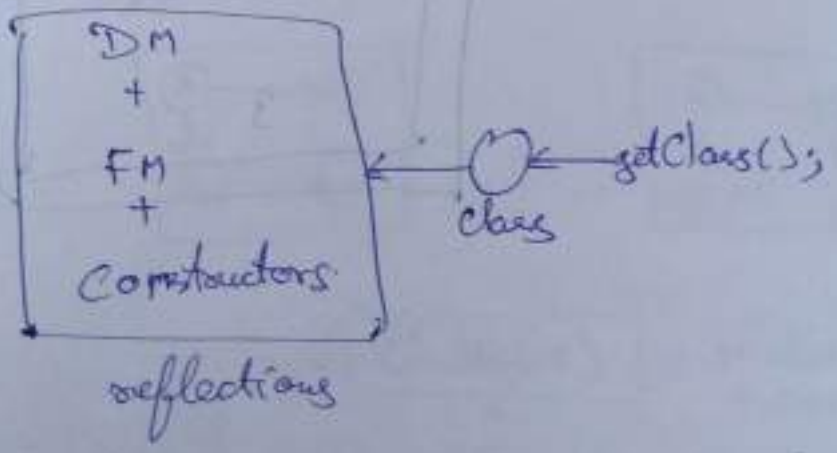
Protected object clone() throws CloneNotSupported Exception.



```
public Student getStudent clone()
{
    Student ref = null;
    try
    {
        ref = (Student) this.clone();
    }
    catch( cloneNotSupported Exception e)
    {
        e.printStackTrace();
    }
    return ref;
}
public void finalize()
{
    System.out.println("I am in finalize....");
}
```

---

reflection (Metadata)



```
Student std1 = new Student(123, "Naveen", 92.0);
Student std = std1.getStudentClone();

    S.o.pln(std1);
    S.o.pln(std2);

    std1 = null;

    System.gc(); // finalize method();

Class c1 = std2.getClass(); // getClass()

    S.o.pln(c1.getName());
    S.o.pln(c1.getSimpleName());
    S.o.pln(c1.getPackageName());
```

String Class (* java.lang Package)

- It is a final class present in java.lang Package
- String is immutable class
- It is Threadsafe
- It's Overloaded Constructors are String(), String("abc"), String(char[] ch)....
- toString(), hashCode() & Equal()

Object class Methods are Overrided.

Immutable class.

```
Public final class A {
    final Private int x = 10
        Public A (int x) {
            this.x = x;
        }
    Public int getx()
    {
        return x;
    }
}
```

```
public class String {
    Psvm (S[] args)
    {
        S.o.pln ("***");
    String s1 = new String();
    String s2 = new String ("abc");
    char[] arr = {'a', 'b', 'c'};
    String s3 = new String (arr);
    s.o.pln (s1);
    s.o.pln (s2);
    s.o.pln (s3);
    S.o.pln (s1.equals (s3));
```

```
s.o.pln(s1.hashcode());
s.o.pln(s2.hashcode());
s.o.pln(s2.hashcode());
```

o/p

```
***
*
abc
abc
0
96354
96354
true
```

-> There are two ways we can Create String Object

1.) by Literals
2.) by new operator

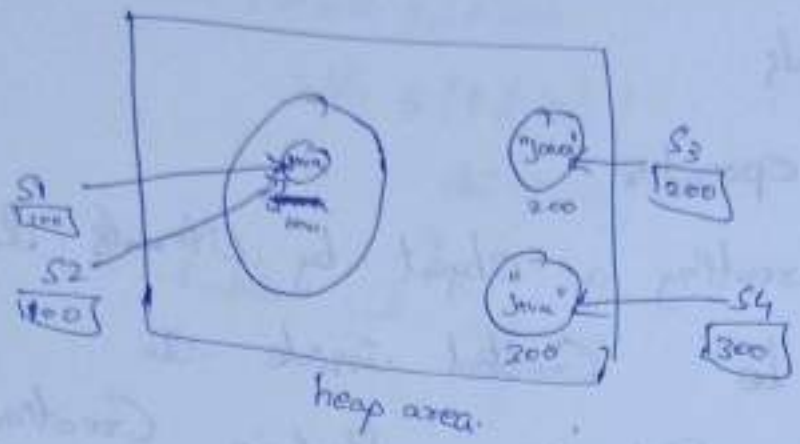
-> If your Creating a object by Literals the object will be Created inside the String Constant Pool Area, Before Creating the object JVM will Checks whether the object value is already Present or not, If it present it won't Create the new object The reference Variable will points to the Same Object

-> If we Create the object by new operator, the object will be Created inside String non-Constant Poll

Why String class is Immutable?

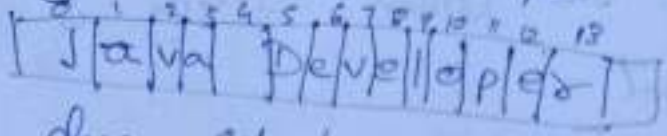
Inside the StringConstantPool a Single object Can be Pointed by Multiple ref. var, if we do the changes with respect to one ref. var it will be reflected in all the

ref. var, Hence they Made String Class as immutable.




```
PSVM (String[] args) {
    String s1 = "java";
    String s2 = "java";   // String Literals.

    String s3 = new String("Java");
    String s4 = new String("Java");   // by new operator

    S.O.pln (s1 == s2); // true
    S.O.pln (s3 == s4); // false
    S.O.pln (s1.equals(s2)); // true
    S.O.pln (s3.equals(s4)); // true
```

String s1 = "Java Developer" s2 = "java" s3 = "Java"



String class will have some many non Static methods.

```
s1.length();//14
s1.charAt(9); // l
s1.indexof('e');//6
s1.lastIndexof('e'); //12
s1.startswith("Java"); //true
s1.endswith("per"); //true
s1.contains("Dev"); // true.
s1.toUpperCase();
s1.toLowerCase();
s1.replace('a','*')
s1.concat("Job");
char[] ch= s1.toCharArray();
s1.split("");
```

```
s2.equals(s3); // false
s2.compareTo(s3); // +ve, -ve, 0
s2.equalsIgnoreCase(s3); // true
s2.compareToIgnoreCase(s3); //0
s1.trim(); // remove leading and trailing Space.
```

abstract method of Comparable interface

```
s1.substring(int beginIndex);//(5) Developer.
s1.substring(int beginIndex, int endIndex); //Develop
```

WAP to check the Given String is Palindrome or not

```
class MC{
    PSVM(S args[]){
        String s1 = "Madam ".toLowerCase().trim();
        String s2 = "";
        for(int i = s1.length()-1; i>=0; i--)
        {
            s2 = s2 + s1.charAt(i);
        }
        if (s1.equals(s2))
            S.O.Pln("Palindrome");
        else
            S.O.pln("Not Palindrome");
    }
}
```

o/p Palindrome

StringBuffer and String Builder

- insert()
- append()
- reverse()
- delete()

STRING	STRING BUFFER.	STRING BUILDER.
1) It is final class Present in java.lang Package	It is a final class present in java.lang Package	It is a final class Present in. java.lang Package
2) It is Immutable	It is Mutable	It is Mutable
3) String it is Thread Safe	It is Thread Safe	It is not Thread Safe
4) In string class toString(), hashCode(), and equals(object o) methods are overrided	Only toString() is overrided	Only toString() is overrided
5) We can Create String object by two ways by literals, by new operator	we can Create the object only by new operator	we can create the object only by new operator
6) String is a Comparable type	String Buffer is Comparable type	StringBuilder is Comparable type

```
class StringDemo {
    Psvm(S a[])
    {
        StringBuffer sb1 = new StringBuffer("abc");
        StringBuffer sb2 = new StringBuffer("abc");
        S.o.pln(sb1); // abc
        S.o.pln(sb2); // abc
        S.o.pln(sb1.hashcode()); // 13652
        S.o.pln(sb2.hashcode()); // 46781
        S.o.pln(sb1.equals(sb2)); // false.
        sb1.append("xyz");
        S.o.pln(sb1); // abcxyz
        sb1.insert("pqr");
        S.o.pln(sb1); // abcpqrxyz
        sb2.reverse();
        S.o.pln(sb2); // cba
    }
}
```

```
S.o.pln("Enter String=");
S.o.pln(new StringBuffer(new Scanner(System.in).next()
    .reverse())
```

```
class main {
    psvm(S a[]){
        String S1 = "mom";
        StringBuffer Sb1 = new StringBuffer(S1);
        Sb1.reverse();
        String S2 = Sb1.toString();
        if (S1.equals(S2))
            S.o.println("Yes");
        else
            S.o.println("No");
    }
}
```

ARRAYS

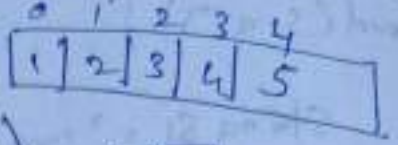
4 ways to Declare and Initialize the Arrays

Syntax

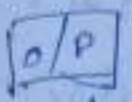
1.) datatype[] Variablename = {v1, v2 .... vn};
2.) datatype Variablename[] = {v1, v2 .... vn};
3.) datatype Variablename[] = new datatype[size];
4.) datatype Variablename[] = new datatype[]{v1, v2 .. vn};

with knowing the length of array

⇒ int[] arr = {1,2,3,4,5};




```
for (int i=0 ; i<=4 ; i++)
{
    S.O.pln(arr[i]);
}
```



1
2
3
4
5

without knowing the length of array

```
// for(int i=0; i<arr.length ; i++)
for(int i=0; i<=arr.length-1 ; i++)
{
    S.O.pln(arr[i]);
}
```

Reverse the Given Array.

```
for(int i = arr.length-1 ; i>=0; i--)
{
    S.O.pln(arr[i]);
}
```

o/p
5
4
3
2
1

Even and odd index elements

```
for(int i=0; i<arr.length ; i++){
    if(i%2==0){
        S.O.pln(arr[i]);
    }
    else if(i%2!=0){
        S.O.pln(arr[i]);
    }
}
```

o/p
10
30
50

o/p
20
40

Enhanced for loop is (For Each)

=> No need of condition and [Inc / dec]

eg- for(int x : arr)
{
s.o.pln(x);
}

(For Each belong to Collections literally As in Arrays also we can use for each loop)

Array -> object
Arrays -> class (object Creation not Possible because Constructor in Private Access)

-> s.o.pln(arr); 14@678 (fully qualified address toString method)
s.o.pln(Arrays.toString(arr)); o/p [1,2,3,4,5]

4 ways access Array

1.) for loop
2.) for each
3.) Index
4.) Arrays class

---

int sum=0;
fo(int x :arr)
{
sum = sum+x
}
s.o.pln(sum);

int e.sum=0, o.sum=0;
for(int i=0; i<arr.length; i++)
{
if(i%2==0){
esum=esum+arr[i];
}
else{
osum=osum+arr[i];
}

NASE

Negative Array Size Exception:

```
public class Student
    {
      int id; String name;  double[] marks;
public  Student (int id, String name, double[] marks)
    {
        this.id=id;
        this.name=name;
        this.marks=marks;
    }
double  totalMarks()
        {
        double total=0.0;
        for (double mark:marks)
        {
          total+=mark;
        }
        return total;
    }
double   averageP()
    {
      return  totalMarks()
    }
String   result()
        {
        String res="PASS";
        for ( double mark:marks)
                {  if (mark<35.0)
                    {
                    res='fail'
                    break; } }
```

```
    return res;
}
public String toString()
{
return "Student [id =" + id + "    ]";
}
}
```

```
Public class Main1 {
PSVM(S[] args) {
Student std1 = new Student[3], "aaa", new double
                [] {100.0, 200, 300};
S.O.pln(std1)
    " (Std1.totalMarks());
    " (Std1.average());
    " (Std.result());
Scanner sc1 = new Scanner(System.in);
S.O.pln("En ");
int id = sc1.nextInt();
S.O.pln("Name");
String name = sc1.next();
double[] marks = new double[3];
for (int i=0 ; i< marks.length; i++)
    {
    Syso("Enter Subject "+(i+1)+"marks");
    marks[i] = sc1.nextDouble();
    }
```

```
Student std2 = new Student(id, name, marks);
S.o.pln(std2);
"  (std2.totalMarks());
"  (std2.average());
"  (std2.result());
}
}
```

```
Public class Main2{
PSVM(){
// int [] arr = {{},{},{}};
//int [][] arr = new int[3][3];
// int[][] arr = new int[][]{{},{},{}};
// int [][]arr = new int[5][6];
// jagged Arrays
int [][] arr = {{}{}};
Syso(arr.length);
Syso(arr[1].length);
for(int i=0; i<arr.length; i++)
{
for(int j=0; j<arr[i].length; j++)
{
S.o.p(arr[i][j]+" ");
}
sysoln();
}
}
```

```
Public class Main {
{
psvm (.) {

String s1 = "program";
char[] arr=s1.toCharArray();
for(int i=0; i<arr.length; i++)
{
for(int j=0; j<arr.length; j++)
{
if (i==j || i+j == arr.length-1)
System.out.print(arr[i] + " ");
else
S.o.p(" ");
}
S.o.pln();
}
}
}
```

```
Public class Main {
{ psvm() {
// String[] arr= {" ", " ", " ", new String{""}};
// String[] arr =new String[3];
// arr[0] ='abc';
// arr[1] = new String ("xyz")
// arr[2] : "pqr";
// String [] arr = new String[] {
}
```

```
Scanner sc1 = new Scanner (System.in)
System.o.pln ("Name Count");
int size = sc1.nextInt();
String[] names = new String[size];
for (int i=0; i<names.length; i++)
{
s.o.pln ("Name" + (i+1) + ":");
names[i] = sc1.next();
}
for (String name : names)
{
s.o.pln (name);
}
```

---

max value in an array.

```
Scanner sc = new Scanner();
s.o.pln ("enter value");
int[] arr = sc.nextInt();
s.o.pln ("Enter numbers");
for (int i=0; i<arr.length; i++)
{
arr[i] = sc.nextInt();
}
int max=0; for (int i=0; i<arr.length; i++)
{
if (arr[i] > max)
{ max = arr[i]; } }
```

```
int [] arr = { 10, 7, 5, 15, 16 }; int fmax = 0, Smax = 0;
for (int i=0; i <= arr.length-1; i++)
  {
    if ( arr[i] > fmax)
      {
        Smax = fmax;
        fmax = arr[i];
      }
    else if (arr[i] > Smax)
      {
        Smax = arr[i];
      }
  }
s.o.pln ("first max" + fmax);
S.O.pln ("first Second max" + Smax);
```

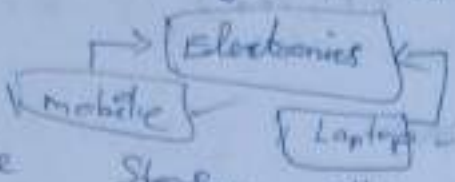
```
int fmin = fmax; Smin = Smax;   // Integer.MAX_VALUE;
for (int i=0; i <= arr.length-1; i++)
  {
    if (arr[i] < fmin)
      {
        Smin = fmin;
        fmin = arr[i];
      }
    else if (arr[i] > Smin)
      {
        Smin = arr[i];
      }
  }
S.O.pln (" first min" + fmin);
S.O.pln (" Second min" + Smin);
```

o/p

first max 16
Second max 15

first min 5
Second min 7

If we Create a Super type Array we can Store Sub class object in it.



while Storing the Sub class object Implicit upcasting will happens

Drawbacks of Array

Fixed in Size => Arrays are fixed in Size
Similar type => Arrays can Store Only homogenious elements
no special methods => There is no Special methods with respect to arrays

Collection

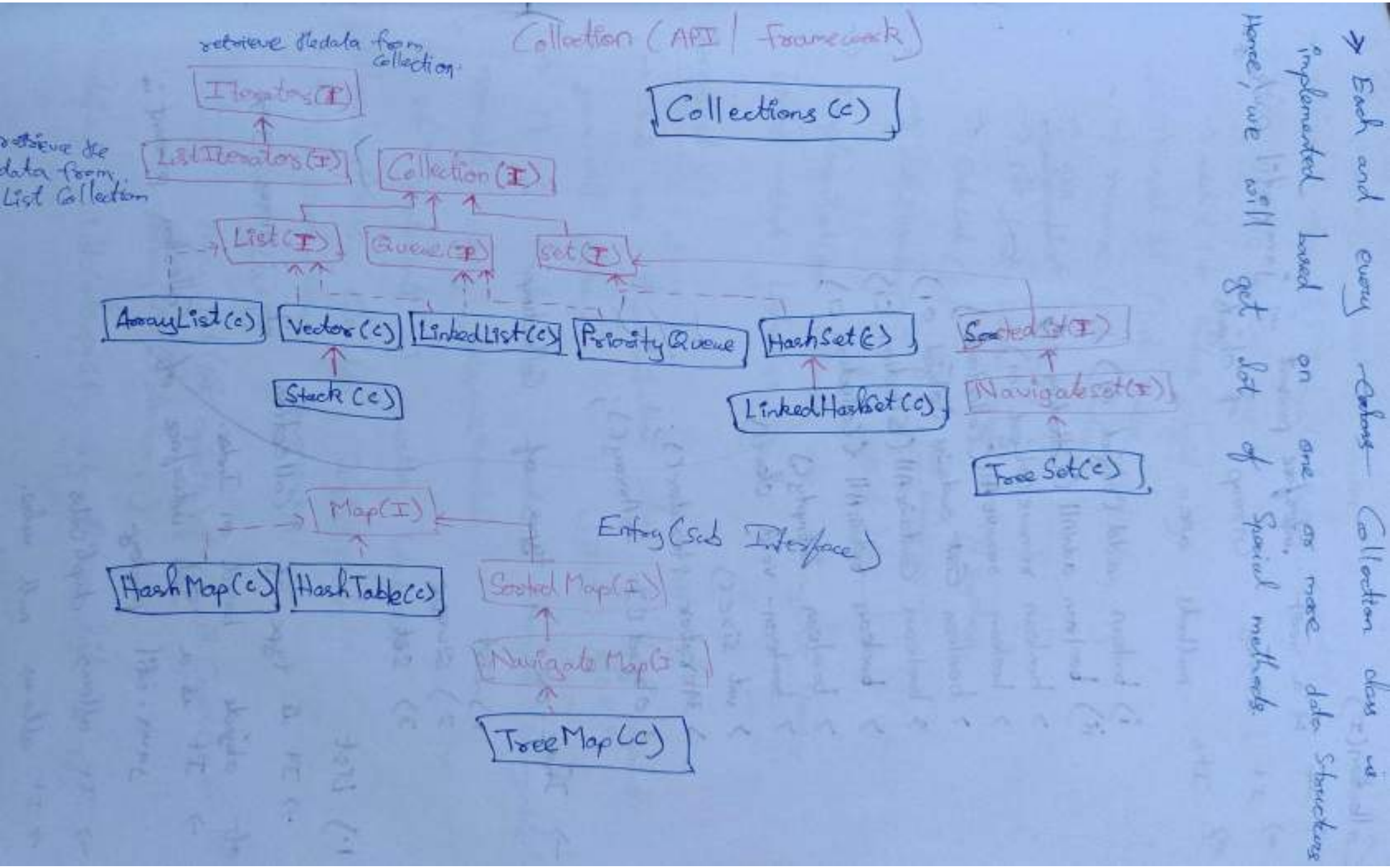
=> Collection is a API or a framework which Stores a Group of object
(application program Interface)

=> Collection framework is Introduced in JDK 1.2.

=> Collection ~~Can Store~~ classes and Interfaces they are present in java.util Package

=> Collection are Dynamic in Nature
Arrays can grow automatically

=> Collections Can Store Homogeneous as well as Heterogeneous elements



Collection(I)

=> It is a root interface present in java.util package.

=> It Stores Group of objects

=> Its methods are

i) boolean add(Object o1)
ii) boolean addAll(Collection c1)
> boolean remove(Object o1)
> boolean removeAll(Collection c1)
> boolean ~~Cont~~ contains(Object o1)
> boolean ContainsAll(Collection c1)
> boolean retainAll(Collection c1)
> boolean isEmpty()
> ~~boolean~~ void clear()
> int size()
> Iterator iterator();
> Object[] toArray();

-> There are 3 types of Collections.

1.) List
2.) Queue
3) Set

1.) List

-> It is type of Collection it Stores group of objects based on Index.
-> It is a Sub interface of Collection present in java.util package.
-> It allows duplicates
-> It allows null value.

-> It preserves insertion order.
-> retrival - foreach / Iterator / index / ListIterator
-> List Specific method are.
    -> add (int index, object o)
    -> addAll (int index, Collection c)
    -> remove (int index);
    -> get (int index);
    -> set (int index, object o) // replace.
    -> subList (int start, int end)
    -> listIterator()
    -> lastIndexOf(3rd)
    -> indexOf(.);
-> Implementation classes are ArrayList, Vector, ~~Linked List~~ LinkedList, Stack.

ArrayList :

-> It is an Implementation class of List present in java.util package.
-> it is implemented based on Dynamic / resizeable / growable array data-structure.
-> It Implements 3 marker interfaces
RandomAccess, Clonable, Serializable.
-> It's overloaded Constructors are
ArrayList()
ArrayList(int capacity) // default Capacity is 10
ArrayList(Collection c)
-> default initialcapacity of Array list is 10.

-> it grows with {newCapacity = oldCapacity * 3/2 + 1}.
-> It is not preferable for insertion/deletion operation
-> we can use Array List for retrival operation

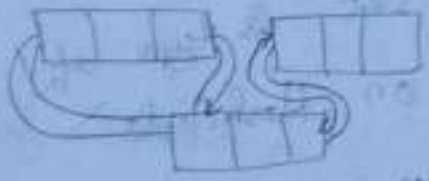
Vector:-

-> It is an Implementation class of List present in java.util package.
-> It is implemented based on Dynamic / resizable / growable array data structure.
-> It implements 3 marker interfaces RandomAccess, Cloneable, Serializable.
-> It is legacy class (It is present from JDK 1.0)
-> Vector class is Thread Safe. (Methods are Synchronized).
-> Its overloaded Constructors are.
Vector()
-> Vector(int capacity)
-> Vector(Collection c)
-> Vector(int initialCapacity, int inc)
-> default initial capacity of Vector is 10.
-> it grows with {newCapacity = oldCapacity * 2}
-> It is not preferable for insertion/deletion operation

-> we can use Vector if we need Thread Safety
-> Its SubClass is Stack (LIFO) push() and pop() methods we can use one Stack.
-> Its methods are addElement(Object o1), removeElement (Object o1), Capacity() .....

LinkedList:-

-> It is an implementation class of List and Queue present in java.util package.
-> It is implemented based on double-linked-list data-Structures.



-> Its overloaded Constructors are
LinkedList()
LinkedList(Collection)
-> no default initialCapacity in LinkedList.
-> it is preferable for insertion/deletion operations
-> don't use LinkedList for Searching Operations.
-> LinkedList Specific methods are
-> addFirst(Object o1), addLast(Object o1), removeFirst(), removeLast(), getFirst(), getLast()

Generics :-

-> It makes Collection as homogeneous.
-> It is given from JDK 1.6.
-> If we use Generics no UpCasting and no need of downcasting

-> Generic Can be used for class not for primitive data types.

-> If we Create Super type Generics we can store Sub class objects

Collection<String> names = new ArrayList<String>(); // jdk 1.5

Collection <String> names = new new ArrayList<>(); // jdk 1.7

Wrapper classes:

Set

Set is type of Collection which stores Object based on HashCode

It is a Sub Interface of Collection present in java.util package.

-> It won't allow duplicates.

-> It allow only one null value.

-> It won't preserve insertion order.

-> retrival can be done by foreach / Iterator

-> We use set for searching operation.

-> no set specific methods

-> Implementation classes are HashSet, LinkedHashSet TreeSet

note:-> if we want to store our objects in Set without any duplicates we must override hashCode() and equals()

HashSet:

It is an implement class of Set present in java.util
It is implemented based on HashTable data-Structure.
HashSet overloaded Constructors are

-> HashSet()
-> HashSet(int capacity)
-> HashSet(Collection c)
-> HashSet(int capacity ~~of hashSet~~, float loadFactor)

-> default capacity of HashSet is 16 and loadFactor (fill ratio) is 0.75 {16*1.0=16 32*1.0=32}

LinkedHashSet:-

-> It is a sub class of HashSet present in java.util.

-> It is implemented based on Hybrid data-Structure [i.e DoublyLinkedList + HashTable].

-> It preserves insertion order.

TreeSet:-

It is an implementation class of set present in java @ .util package.

(*) → It stores objects in sorted order.

(*) → TreeSet Implemented based on Balanced Tree data structure

(*) → It stores only Comparable type objects

(*) → It stores Homogeneous elements.

→ It won't allow duplicates, null values and won't preserves insertion order

→ Retrival by foreach | Iterator.

→ It's overloaded Constructors are-

→ TreeSet ()

→ TreeSet (Collection c)

→ TreeSet (SortedSet s)

→ TreeSet (Comparator c).

Diff btw Comparable and Comparator

Comparable is an interface present in java.lang

Comparable / Comparator is an interface present in java.util.

Comparable abstract methods

i) public int CompareTo(Object o1)

Comparator ii) public int Compare (Object o1, Object o2)

Collection Stores Object in sorted order based on Comparable implementation.

If we want to sort objects based on some other factors we must use Comparator.
Customized Sorting

Priority Queue :-

->It is an Implementation class of Queue present in java.util.

→ It Stores objects in natural ascending order
→ It Stores only Comparable type objects
*→ It allows duplicates
→ It won't allow null Value
→ It can't preserve insertion order
→ retrival → foreach / Iterator [retrival is random]
→ pak() and poll() we can use to retrive elements in Sorted order.

Map

It is an interface present in java.util package
It Stores object in the form of key and Value pair
Key and Value both are objects
key can't be duplicate but Value Can be duplicate

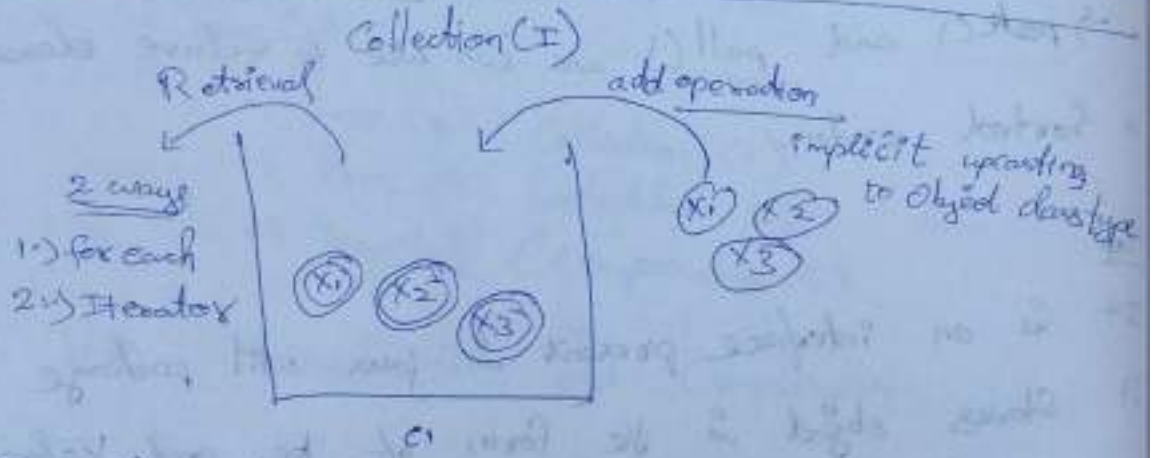
→ Put (Object key, Object value)
PutAll (Map m);
remove (Object key)
containsKey (Object key)
Contains Value (Object Value)
get (Object key), keySet(), Values(), entrySet()
Size(), isEmpty(), clear()

Non Static & method
setKey()
getValue()

Implementation class HashMap, HashTable, TreeMap

Diff b/w HashMap and HashTable

HashMap	HashTable
key and value can be null	key and value can't null.
Is not legacy class	It is a Legacy class
It is not ThreadSafe	It is ThreadSafe
Faster in Performance	Slower in Performance



1.) for each

→ Enhanced for-loop

→ Given from Jdk 1.5

```
for (Object o1 : C1)
{
  // Downcast if use
}
```

2) Iterator (I) — ListIterator (I)

Object next();
boolean hasNext(); } abstract methods
void remove();

previous(), hasPrevious

Iterator itr = c1.iterator() => while(itr.hasNext());

Object o1 = itr.next();
Object o2 = itr.next();
Object o3 = itr.next();
Object o4 = itr.next();
Object o5 = itr.next(); // NoSuchElementFound Exception

{
Object o1 = itr.next();
// Downcasts o1 to use
}

Object next(); will move to next object

boolean hasNext(); will check and return if the object is there or not

void remove(); will remove the pointed object

---

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
Student class
public class MainClass {
psvm {

Collection c1 = new ArrayList();
c1.add("abc");
c1.add(new String("xyz"));
c1.add(new StringBuffer("56"));

```
c1.add(new pen());
c1.add(new Student(123, "Student1", 100.0));
//c1.clear();
Syso(c1.size());
Syso(c1.isEmpty());
Syso(c1.contains("abc"));
```

```
for(Object o1:c1)
    {
    syso(o1);
    }
```

```
Iterator itr = c1.iterator()
  while(itr.hasNext())
      {
        Object o1 = itr.next();
        syso(o1);
      }
  }
}
```

---

olp

```
5
False
True
abc
xyz
sb
Collectiondemo.pen@14A61
123,Student1,100.0
abc
xyz
sb
Collectiondemo.pen@14A61
123,Student2,100.0
```

Without Generic

```
Collection names = new LinkedList();

names.add("xyz");
names.add("wxy");

for(Object o1 : names)
{
String name = (String) o1; //downcasting
sysO (name + 'lt' + name.charAt(0) + "lt" + name.length());
}
```

With Generic

```
Collection<String> names = new LinkedList<String>(); //jdk1.6

names.add("xyz");
names.add("wxy");

for (object o1 : names)
  {
    Syso(o1 + o1.charA(0) + o1.length());
  }
```

-> `Collection<String> names = new LinkedList<>();` //jdk1.7

WRAPPER CLASS

boolean -> Boolean
char -> Character
byte -> Byte
short -> Short
int -> Integer
long -> Long
double -> Double
float -> Float

i) are final class present in java.lang package.
ii) are immutable
iii) are having Overloaded Constructors
iv) toString(), hashCode() & equals() methods are overrided.
v) are Comparable type

1.) * AUTO BOXING
Converting Primitive Datatype to wrapper class
eg- Integer i2 = 50;

2.) * AUTO UNBOXING
Converting wrapper class type to Primitive Datatype
eg => int x = i2;

3.) * Parsing
String to int using Static methods
eg:- String s1 = "1234";
int y = Integer.parseInt(s1);

Converting String Data to Primitive type

NOTE => Can We Store Primitive data in Collection => NO
Collection Can Store Only object, if we try to Store primitive in Collection it will Autoboxing to Respective wrapper class file

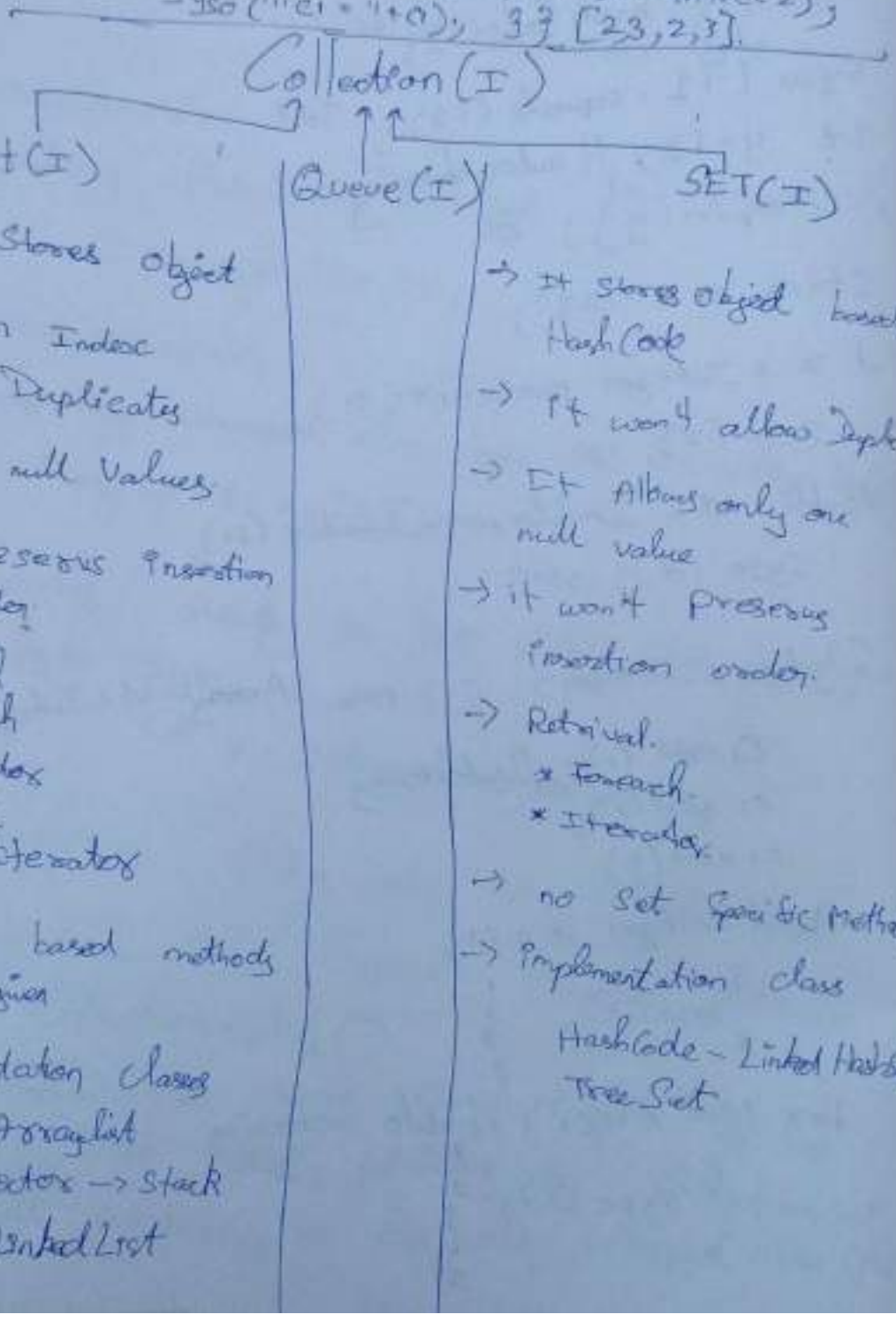
```
public class MainClass {
    psvm() {
    Integer i1 = new Integer(50);
    Integer i2 = 50; // autoboxing.
    Syso(i1);   50
    Syso(i2);   50

    Syso(i1.hashCode());  50
    Syso(i2.hashCode());  50

    Syso(i1.equals(i2));  True
    int y=i1; // autounboxing
      S.O.println(y);  50
    String s1 = "123";
    int z = Integer.parseInt(s1);
      Syso(z);  123
    double d = Double.parseDouble(s1);
      Syso(d);  123.0

    Collection<Integer> c1 = new ArrayList<Integer>();
      c1.add(1); //autoboxing
      c1.add(2);
      c1.add(3)
    for(Integer x : c1)
      { Syso(x);   1
    }              2
                   3
                   4
                   5
    for(int x : c1) // auto unboxing.
      { Syso(x);   1
    }              2
                   3
                   4
                   5
```

Syso ("c1=" + c1), (1, 2, 3, 4, 5, 2, 3)

Collection <Integer> c2 = new ArrayList <>();

c2.add(2);
c2.add(3);

c1.addAll(c2);
Syso("c1=" + c1);
c2.add(7); // Syso(c1.containsAll(c2)); T
// c1.removeAll(c2); c1.retainAll(c2);
Syso("c1 = " + c1); // [2,3,2,3].



Exception Handling

> Exception is an unexpected event which makes program to terminate abnormally
> there are 3 types of Termination.
1.) Normal Termination.
2.) Force Termination. (System.exit(0))
3.) Abnormal Termination. (Exception)
> Even there is an Exception we can continue execution by Handling Exception.
> We can handle Exception by using try-catch
> In try we must write the code which causes the Exception.
> In catch we must have solution code.
> try must be followed by Catch or finally.
> Catch block will executes only when exception in try
> Finally block will executes always.
> If try followed by Catch program will continue the exception.
> If try is followed by finally will executes but program will terminates abnormally
> (If we do Force Termination finally will not executes.)

Diff b/w Error and Exception

Error

> Error is class present in java.lang
> Error will occur because of lack of System resources.
> Error are Irrevokable
Ex:- for Error:
StackOverflowError,...
NoClassDefFoundError,...

Exception

> Exception is class present in java.lang
> Exception will occurs because of unExpected Event.
> Exception are Revokable
Ex:- for Exception:
ArithmeticException
CloneNotSupportedException...

Diff b/w Checked Exception and Unchecked Exception:

Checked Exception

> Exception Identified by Compiler is known as Checked Exception.
> All checked Exceptions are Sub class of Exceptions

Unchecked Exception.

> Exception NotIdentified by Compiler is known as Unchecked Exception.
> All unchecked Exception are sub class of Run time Exception.

Types of Exception

1.) Checked / Caught / Compiletime Exception
2.) Unchecked / unCaught / RunTime Exception.

Try with Multiple Catch blocks:

> a try can have multiple Catch but Only one finally.
> Only one Catch block will be executed
> Multiple Exception can be handled in same Catch. (from jdk 1.7)
> Super class Catch can be handled both checked / Unchecked exception.
> Catch (Exception) Can Handle both checked / Unchecked exceptions.
> Catch can be written up to Throwable.
> while defining Multiple Catch we should have from Sub class to Super class

Default Exception Handling in JAVA

-> If we are not Handling Exception JVM will Handle the exception
-> If JVM is Handling Exception it will Terminate the program execution and also prints stackTrace.
-> we can print Stack Trace by using printStackTrace().

How to Create Usersdefined / Customized Exceptions?

→ by extending any of the Exception class.

throws and throw

→ throw is a keyword which is used to throw an object which is of type Throwable

→ throw is generally used with Userdefined / Customized Exceptions

→ we can throw only one Obj at a time.

→ throw must be used in definition block.

→ Syntax :- throw new ClassName();

---

throws

→ throws is a keyword which is used to pass the exception to Caller.

→ throws is Generally used with Checked Exceptions

→ throws can be used to pass multiple Exceptions to Caller.

→ throws can be used in Declaration part

→ Syntax :- throws className1, ClassName2....

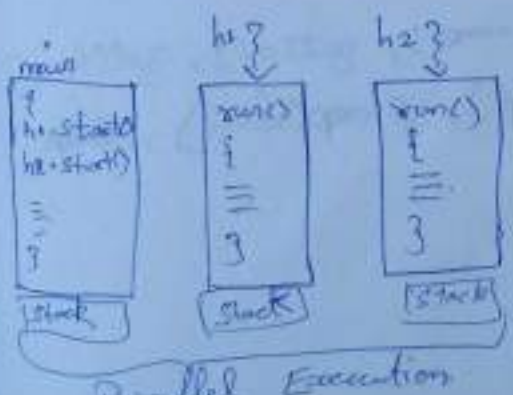
Threads

-> Threads is an seperate execution Instance. It will have it's own path of Execution
-> Threads is a light weight Process
-> Threads are used for Parallel Execution
-> Java Supports multi threading
-> In java we can Create Threads in 2 ways
    1.) Extending from Thread class.
    2.) implementing from Runnable interface.

Diff b/w Start() and run()

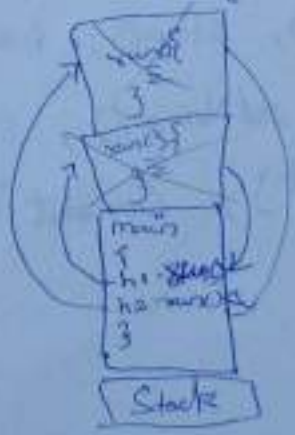
Start()

> It is a nonstatic method of Thread class.
> If we call Start() execution is parallel
> If we call Start() is on different Stack frame



run()

> It is a abstract method of Runable interface
> If we call run() execution is Sequential
> If we call run() execution is on same stack frame.



Sequential Execution

Types of Thread:

1.) User Threads:
2.) System Threads:

* Thread Scheduler, (Allocating Resources) to particular thread
* Garbage Collector, (Low Priority) (destroy)
* Thread main (executes the main method)

Thread class:-

> Thread is a class present in java.lang.
> Thread is having overloaded Constructors

Thread()
Thread(String name)
Thread(Runnable r)
Thread(Runnable r, String name)

> Thread properties are

Id → defaultly given by JVM
Name → defaultly given by JVM.
Priority → is b/w 1-10 and default is 5

> Its methods are:-

start(), stop(), pause(), resume(), getId(), getName(), getPriority(), setName(), setPriority(), sleep(long), sleep(long, int), currentThread()

Thread Safety

> If multiple thread modifying object it leads to data inconsistency
> Instead of allowing multiple Threads to access objects we will allow only one thread to access Object this is known as thread Safety
> 2 ways we can achieve thread Safety
    1. develop methods with Synchronized.
    2. develop ~~methods~~ Class as immutable.
> Multiple Threads are waiting for same thread resource Infinitely then it is known as deadlock
> We can slove deadlock by Inter Thread Communication methods
    → wait()
    → notify()
    → notifyAll()

Diff b/w Iterator and ListIterator

Iterator

> It is an Interface present in java.util package

> It is used to retrieve the elements of Collections

> If we Create Iterator object the Cursor can Travel only in forward direction by using i) next();
ii) hasNext();

> We can Create Iterator object by using iterator() methods present Collection Interface

ListIterator

> It is Sub Interface of Iterator present in java.util package.

> It is used to retrieve the elements of List type of Collection

> If we Create ListIterator the obj the Cursor can Travel both forward and Backward direction by using
i) previous();
ii) hasPrevious();

> we can Create ListIterator object by using listIterator() method present List(I) Interface

↑

```
import java.util.ArrayList, LinkedList, List, ListIterator
                                                    ,Vector;
public class MainClass{
    psvm(String[] args)
    {
    // List<Integer> nums = new ArrayList<Integer>();
    // List<Integer> nums = new Vector<Integer>();
    List<Integer> nums = new LinkedList<Integer>();
    nums.add(22);0
    nums.add(11); 1
    nums.add(45);
    nums.add(34);
    nums.add(1);
  // index based Operation
  // Preference will be given to Index
  Syso(nums); [22, 11, 45, 34, 1] 34.
                 2       4
  // nums.remove(2);
  nums.set(2,42); //Replace
  nums.add(1,33);
  for(int i=0; i<nums.size(); i++)
    {
      syso(i+"--->"+nums.get(i));
    }
    Syso(nums.subList(1,4));
    Syso(nums.indexOf(34));
    Syso(nums.lastIndexOf(34));
```

```
//ListIterator itr = nums.listIterator();
ListIterator itr = nums.listIterator(3);
while (itr.hasNext())
    {
      System.out.println(itr.next());
    }
  while (itr.hasPrevious())
    {
      syso(itr.previous());
    }
  }
}

Iterator itr = cr.iterator();
```

```
import java.util.HashSet; LinkedHashSet; Set
public class MC6 {
    psvm(String[] args)
    {
    Set names = new HashSet();
//Set names = new LinkedHashSet();
names.add("Disha");
names.add("silk");
names.add("Sam");
names.add(new String("xyz"));
names.add(new String("XYZ"));
names.add(null);
names.add(new StringBuffer("ABC"));
names.add(new StringBuffer("ABC"));
for(Object o1 : names)
{
    Syso(o1);
}
}
}
```

O/P

null
ABC
Disha
ABC
Sam
xyz
XYZ
silk

```
public class Employee {
    psvm() {
    int Empid;
    int String name;
      double Sal;
  Employee() { Int Empid, String name, double sal)
        this.Empid = Empid;
        this.names = names;
       this. Sal = Sal;
    }
Public String toString()
    {
      return "Employee [id =" +id + ", name=
            + name + ", Salary=" +
                    Salary + "]";
    }
Public int hashCode()
    {
      return id;
    }
public boolean equals(Object o,)
    {
  return this.hashCode() == o. hashCode();
    }
    }}
```

```
public class MC7 {
    psvm() {
    Set<Employee> s1 = new HashSet();
s1
public int compareTo(Object o)
    {
    return this.hashCode() - o.hashCode(); //ascending
    //return o.hashCode() - this.hashCode(); //descending
    // Employee ref = (Employee) o;
    // if (this.Salary > ref.Salary) return 5;
    // else if (this.Salary < ref.Salary) return -5;
    // else return 0;
    // return this.name.compareTo(ref.name);
    }
}
}
```

```
import java.util.Set;
public class MC7
{
    PSVM() {
    Set<Employee> emps = new HashSet<Employee>();
    emps.add(new Employee(123, "ABC", 20000));
    emps.add(new Employee(345, "DEF", 3000.0));
    emp.add(new Employee(467, "GEF", 4000.0));
    emp.add(new Employee(123, "ABC", 2000.0));
    for(Employee e1:emps)
    {
        sysO(e1);
    }
}
```

o/p

Employee [id=123, name=ABC, Salary=2
Employee [id=345, name=DEF, Salary=3
Employee [id=467, name=GEF, Salary=4

```
import java.util.Set;
import java.util.TreeSet;
public class MC {
    psvm(String[] args)
    {
        Set ts = new TreeSet();
        ts.add(10);
        ts.add(8);
        ts.add(12);
    // ts.add("abc"); // ClassCastException;
    // ts.add(null); // NullPointerException;
    sysO(ts);
    }
}
```

o/p

~~8~~
~~10~~
12

o/p

[8, 10, 12]

```
import java.util.TreeSet;
Public class MC {
    PSVM(String args[]) {
    Set<String> names = new TreeSet<String>();
    names.add("ABC");
    names.add("VMP");
    names.add("ABK");
    names.add("ASBJ");
    sysO(names);
    sysO("ABC".compareTo("ASBJ"));
}
}
```

o/p

[ABC, ABK, ASBJ, VMP]
28

returns 0, +ve, -ve

```
import java.util.TreeSet;
public class MC {
    PSVM (S args[]) {
        Set<Employee> emps = new TreeSet<>();
        emps.add (new Employee (10, "ABC", 20000.0));
        emps.add (new Employee (12, "XYZ", 4000.0));
        emps.add (new Employee (8, "AAB", 1000.0));
        for (Employee e1 : emps)
        {
            Syso (e1)
        }
    }
}
```

o/p

```
8   AAB   1000.0
10  ABC   20000
12  XYZ   4000.0
```

```
import java.util.TreeSet;
public class MC {
    PSVM (SC] args) {
        Set<Employee> emps1 = new TreeSet<>();
        emps1.add (new Employee (10, "ABC", 20000.0));
        emps1.add (new Employee (12, "XYZ", 4000.0));
        emps1.add (new Employee (8, "AAB", 1000.0));
```

```
for(Employee e1 : emps1)
    { sysso(e1);
    }

SortBySalary sbs = new SortBySalary();
Set<Employee> emps2 = new TreeSet<>(sbs);
emps2.add(new Employee(10, "ABC", 2000.0));
emps2.add(new Employee(12, "XYZ", 4000.0));
emps2.add(new Employee(8, "AAB", 1000.0));
emps2.addAll(emps1); for(Employee e1 : emps2)
                        { sysso(e1);
                        }
}

class SortBySalary implements Comparator
{
    public int compare(object o1, object o2)
    {
        Employee e1 = (Employee)o1;
        Employee e2 = (Employee)o2;
        if (e1.Salary > e2.Salary) return 5;
        else if (e1.Salary < e2.Salary) return -5;
        else return 0;
    }
}
}
```

op

```
8   AAB  1000.0
10  ABC  2000.0
12  XYZ  4000.0

12  XYZ  4000.0
10  ABC  2000.0
8   AAB  1000.0
```

pre-order (Root, L, R)
Post-order (L, R, Root)
In-order (L, Root, R) ✓ (Tree Set)

Comparable will return (0, +ve, -ve)

Map

1) key-Value
2) No duplicate key

- HashMap : UnOrdered
- TreeMap : Sorted
- LinkedHashMap : Insertion ord
- Hashtable : Synchronized

i) Set<String> ks = mobiles.keySet()

Set@1000

"1100"	"Samsung"	"Apple"	"oppo"

Entry → key-Value

key	value
1100	15
Apple	3
Samsung	8
Oppo	12

ii) Set<Entry<String, Integer>> es = mobiles.entrySet()

SET@100

K	V
1100	15
S	8
A	3
opp	12

```
public class HM1
  psvm(String[] args) {
   HashMap<String, Integer> mobile = new HashMap<>();
   mobiles.put ("1100", 15);
   mobiles.put ("Apple", 3);
   mobiles.put ("Samsung", 8);
   mobiles.put ("Oppo", 12);
   syso (mobiles);
// Searching key
   syso (mobiles.containsKey ("Apples")); // true.
// Removing an Entry from map
   mobiles.remove ("1100");
// Accessing using set of keys
  Set<String> ks = mobiles.keySet();
  for (String mobile : ks)
   {
     if (mobiles.get (mobile) >100)
        {
          syso(mobile);
   }    }
// Accessing using set of Entries
  Set<Entry<String, Integer>> es = mobiles.entrySet();
   for(Entry<String, Integer> entry : es)
        { if (entry.getValue()>10) {
   syso (entry.getKey()); } } }
```

WAP to print the Frequency of characters in the given String.

```
P c PrintDuplicate {
    Psvm (String[] args)
    {
      String str = "Programmer";
      LinkedHashMap<Character, Integer>
                        fv = new LinkedHashMap<>();
    for (char ch : str.toCharArray())
    {
      if (fv.containsKey(ch))
        {
          fv.put(ch, fv.get(ch)+1);
        }
      else
        {
          fv.put(ch, 1);
    }  }
    syso (fv);
    for(Entry<Character, Integer> entry: fv.entry
        { syso(entry);
        }
    }
}
```

Priority Queue

Program:-

```
p e mc {
   psum ( s a[] )
     {
     Queue <Integer> pq = new priorityQueue <Integer>();
        pq.add(8);
        pq.add(7);
        pq.add(1);
        pq.add(4);
        pq.add(9);
        pq.add(3);
   // S.o.pln(pq); // (1,3,4,7,8,9)
   // sysо(pq.peek()); -> gives head element and not
   // syso(pq);                    remove from the Collection.
   // sysо(pq.poll()); -> gives head element and remove
   // syso(pq);                    from the Collection.
     Object o1 = pq;
      while (o1 != null)
           {syso(o1);
              o1 = pq.poll();
           }
         syso(pq); //null
     }
   }
```

Exception Handling

eg:-

```
P c MC {
    PSVM (s) {
        Syso ("* * * *");
        int n1 = 10;
        int n2 = 0;
        int n3 = 0;
        try
        {
            // code Causes Exception
            n3 = n1/n2;
        }
        Catch ( Exception e1 )
            {
                // Solution code
                n3 = n1/1;
            }
        Syso ("n1 value =" + n1);
        Syso ("n2 value =" + n2);
        Syso ("n3 value =" + n3);
    }
}
```

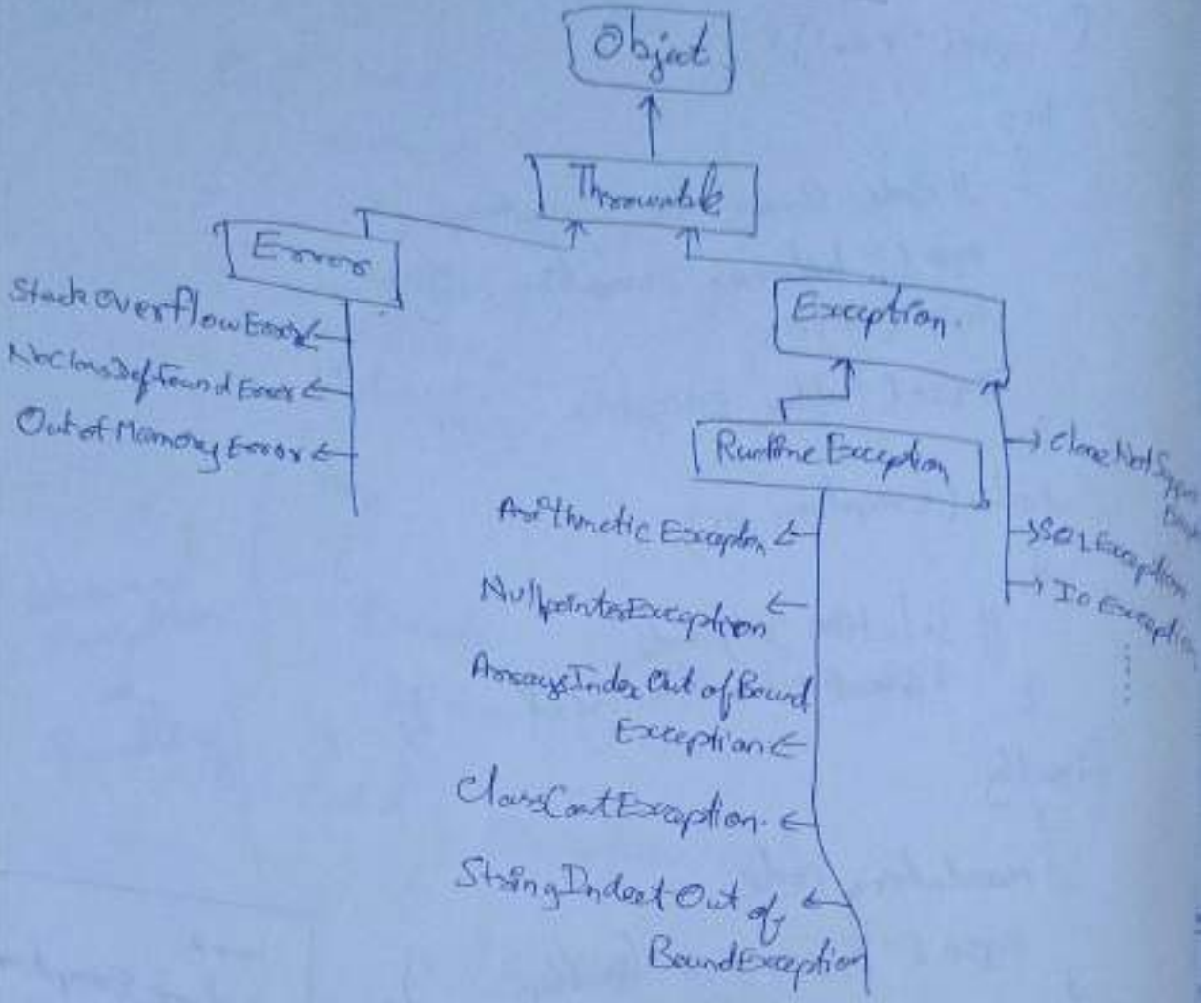
abnormal Exception is Handling

```
* * * *
n1 value = 10
n2 value = 0
n3 value = 1
```

```
PCM {
    psvm(String[] args)
    {
      syso("...");
      try
      {
        // Code Causes the Exception.
        syso("before Exception...");
        syso(10/0);
        syso("after Exception...");
      }
    Catch (Exception e1)
      {
        // solution of code
        syso("I am in Catch...");
      }
    finally
      {
      // mandatory code
        syso("I am in finally....");
      }
    }
}
```

```
...
before Exception
I am in Catch
I am in finally
```

Exception Handling



Try with Multiple Catch Blocks

```
P C MC{
    PSVM(){
        Syso("----");
        try
        {
            String s1 = "abc";
            Syso(10/2);
            Syso(s1.charAt(10));
        }
        Catch(ArithmeticException / NullPointerException e)
        {
            Syso("AE/NPE");
        }
```

```
Catch (RuntimeException e1)
  {
   syso("RE");
  }
Catch (Exception e1)
  {
   syso("Exception");
  }
 syso("-----");
 }
}
```

① default Exception Handling in JAVA

eg:-
```
PC mc{
    Psvm(){
     syso("-----");
     m1();
     syso("-----");
    }
 static void m1(){
     m2();
    }
 static void m2(){
     syso(10/0);
    }
}
```

throw and throws

Eg:-

```
class InvalidAmountException extends Exception
{
    InvalidAmountException()
    {
      syso("IAE object is Created...!!");
    }
    void amountHandling()
    {
      syso("GetLost you culprit...!!!");
    }
}
Public class BankApp
{
    Psvm(String[] args)
    {
      initiateTXN(-1000.0);
    }
    private static void initiateTXN
                        (double amt)
    {
      try
      {
        transaction(amt);
      }
      Catch(InvalidAmountException e)
      { e.amountHanding(); e.PrintStackTrace
```

```
    }
}
private static void transaction(double amt)
                    throws InvalidAmountException
{
  if (amt>0)
    {
      sysO("Transaction is Initiated for "+amt+"$");
    }
  else
    {
      throw new InvalidAmountException();
    }
}
```

Threads (Execution instance)

ways to Create Threads

- Extending from Thread
- implement from Runnable Interface

⇒ Java will support multiple threading

abstract Run() Method.

eg:-

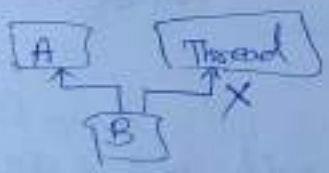
```
class Hero extends Thread
{
    public void run()
    {
    for(int i=1; i<=10; i++)
    {
    sysо("i value = "+i);
    try
    {
      Thread.sleep(1000);
    }
    Catch (InterruptedExeception e)
    {
      e.printStackTrace();
    }
  }
 }
}
class Demon implements Runnable
{
    public void run()
    {
      for(int i=100; i<=110; i++)
      {
      sysо("i Value=" + i);
      try
      {
        Thread.sleep(1000);
      }
      Catch(InterruptedException e)
      {
        e.printStackTrace();
      }
}}}
```

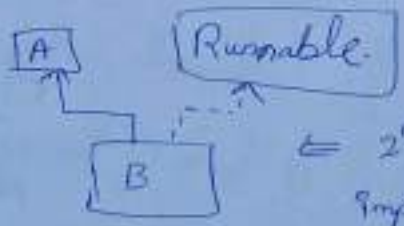
```
public class MC {
    psvm () {
        Hero h1 = new Hero();
        Hero h2 = new Hero();
            h1.start();
            h2.start();
        Demon d1 = new Demon();
        Demon d2 = new Demon();
            // d1.run();
            // d2.run();
        Thread t1 = new Thread(d1);
        Thread t2 = new Thread(d2);
            t1.Start();
            t2.Start();
    }
}
```

---

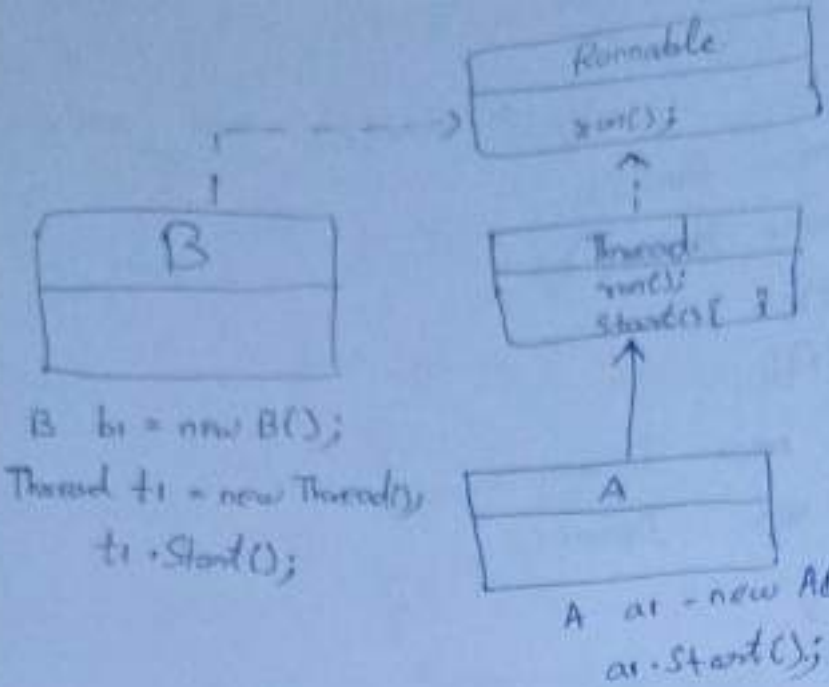
why we have two ways to have threads.

i) It won't Support multiple Inheritance.





⇐ 2nd ways implements from Runnable.



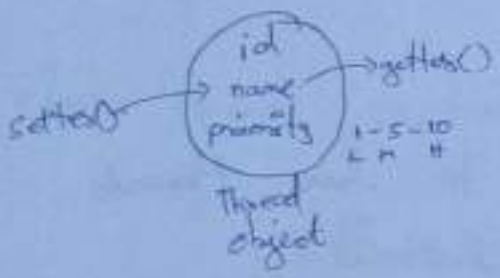
B b1 = new B();
Thread t1 = new Thread();
t1.start();

A a1 = new A();
a1.start();

Type of Threads

User Threads

System Threads



- Thread Scheduler (Allocates resources among the threads)
- Garbage Collector (Low priority)
- Thread main

↓

Executes the main method.

```
Class A
{
public static void main (String[]
```

Hardcoded in Thread main

```
{
}
}
```

> java A 10 20 abc (command arguments)

'10'	'20'	'abc'

size 3

> java A

size 0

After class we can pass arguments to String Array

```
package Threads;
p c mc {

//Public Static void main (String[] args)
//public static void main (String[] args) //error
// public void main (String[] args) // error
// public Static int main (String[] args)// error
//public Static int Main (String[] args)// error
//public Static void main (int[] args)// error
//public static void main (String[ args) //works

// we can't define main as abstract because it is
static.
// Static public void main (String[] args) //works
// we can Overload main() Method.
// we can't overide main() method.
// psvm (String... args[])// works
psvm()
{
// it gives currently executing Thread
Thread t1 = Thread.currentThread();
Syso (t1.getId()); // 1
Syso (t1.getName()); // main
Syso (t1.getPriority()); // 5
Syso (args.length); // 3 (command line)
Syso (args[0] + args[1]); // 1020
int x = Integer.parseInt (args[0]);
int y = Integer.parseInt (args[1]);
Syso (x+y); // 30 } }
```

```
package threads;
class MyThread extends Thread
{
  public void run()
  {
    Thread t1 = Thread.currentThread();
    for(int i=1; i<=100; i++)
    {
      sysо(t1.getName());
    }
  }
}
Public class MC3
{
  PSVm (String args[]) {
    Thread t1 = Thread.currentThread();
    MyThread mt1=new MyThread();
      mt1.setName('child');
      mt1.setPriority(1);
      mt1.start();
      for(int i=1; i<=100; i++)
      {
        syso("parent = "+t1.getName());
      }
  }
}
```

```
class Greeting
{ public Synchronized static void greetA
  {
    String data = "Good Afternoon";
    for (int i=0; i<data.length(); i++)
    {
      sysout (data.charAt(i));
      try {
        Thread.Sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    sysoln();
  }
class GreatThread extends Thread {
  { public void run()
    {
      Greeting.greetA();
    }
  }
```

```
public class MainClass
{
    Psvm(String[] args)
    {
        GreatThread gt1 = new GreatThread();
        GreatThread gt2 = new GreatThread();
        GreatThread gt2 = new GreatThread();
        gt1.start();
        gt2.start();
        gt2.start();
    }
}
```

Diff b/w Sleep & Wait

Sleep

-> Sleep is a static method of Thread class

-> If we call sleep method the Thread will Hold the lock and goes to blocked state

Syntax

-> public var static void sleep(long ms);

-> public static void sleep(long ms, int ns);

Wait

-> Wait is non static method of Object class

-> If a Thread execute the wait method it releases the lock and goes to blocked State

Syntax

public void wait();

public void wait(long ms);

public void wait(long ms, int ns);

NOTE :-

=> Both the methods will throws InterruptedException

Diff blw Synchronized keyword and Synchronized block

Synchronized keyword

If we define any method with Synchronized keyword the Complete method Content will becomes thread Safe

eg:-

```
Synchronized void test() {
    ≡≡
}
```

(Synchronized method)

Synchronized block

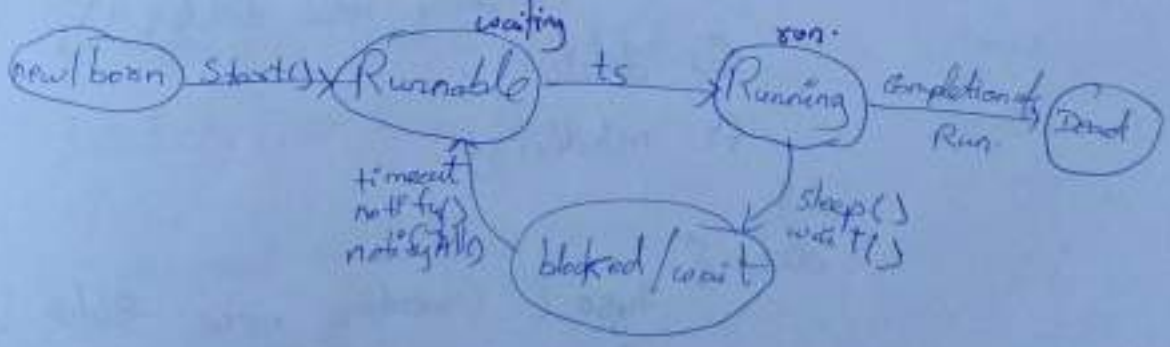
If we do want to make a block of code need to be thread Safe we can use Synchronized block.

eg:-

```
void test()
{
    ≡≡
    Synchronized(this)
    {
        ≡≡
    }
    ≡≡
}
```

(Synchronized block)

THREAD STATE DIAGRAM



```
Package threads;
class Tickets
{
    int availableTickets = 2;
    Synchronized void bookTicket()
    {
        String name=
```

File Handling

to Create a Folder.

```
import java.io.file;
public class MC1
{ psvm(S[] args) {
        syso ("****");
        file f1=new file ("c:\\users\\");
                                  path
        if (f1.exists())
        {
            syso ("Deleting old folder and
                   Creating new folder");
            f1.delete();
            f1.mkdir();
        }
        else {
            syso ("Creating new Folder..!!");
            f1.mkdir();
        }
```

```
        Syso("****");
    }
}
```

o/p

Deleting old folder and
Creating new folder.

To Create file

```
import java.io.File;
import java.IOException;
public class mc2 {
    psvm(SC[] args) {
        Syso("****");
        File f1 = new File("Path\\filename.txt");
        try
        {
        f1.CreateNewFile();
        Syso(f1.canRead());
        Syso(f1.canWrite());
        Syso(f1.canExecute());
        Syso(f1.getAbsolutePath());
        Syso(f1.getName());
        Syso(f1.setWritable(false));
    } Catch (IOException e)
    {
      e.printStackTrace();
    }
    Syso("done");
    Syso("****");  } }
```

o/p

```
true
true
true.
Path
filename
true
done.
```

To write in file

```
import java.io.file;
import java.io.filewriter;
import java.io.IOException;
public class MC3 {
P SVM (SC] args[].
  {
   Syso("****");
  file f1 = new file("file Path");
  Filewriter fw = null;
  try
   {
   f1.setWritable(false true);
   fw = new filewriter(f1);
   fw.write("file Handling \n");
   fw.write("file Handling");
   fw.flash();
   syso("done");
  } catch (IOException e) {
       e.printStackTrace();
  }
   finally
    {
      if (fw != null)
       {
        try {
            fw.close();
       }
```

```
catch (IOException e) {
    e.printStackTrace();
  }
 }
}
 syso("***");
 }
}
```

O/P
* * * *
done
* * *

To Read a file.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class MC4
{
 psvm (String[] args)
 {
  File f1 = new File("file path");
  FileReader fr=null;
 try
 {
  fr=new FileReader(f1);
  int size =(int) f1.length();
  char[] arr = new char[size];
```

```
            fr.read(arr);
            String data = new String(arr);
            System.out(data);
        } catch (IOException e)
            {
                e.printStackTrace();
            }
        finally
            {
                if (fr != null)
                {
                    try {
                        fr.close();
                    } Catch (IOException e){
                        e.printStackTrace();
                    }
                }
            }
```

BufferReader => used to Read line by line instead of char by char

To read line by line

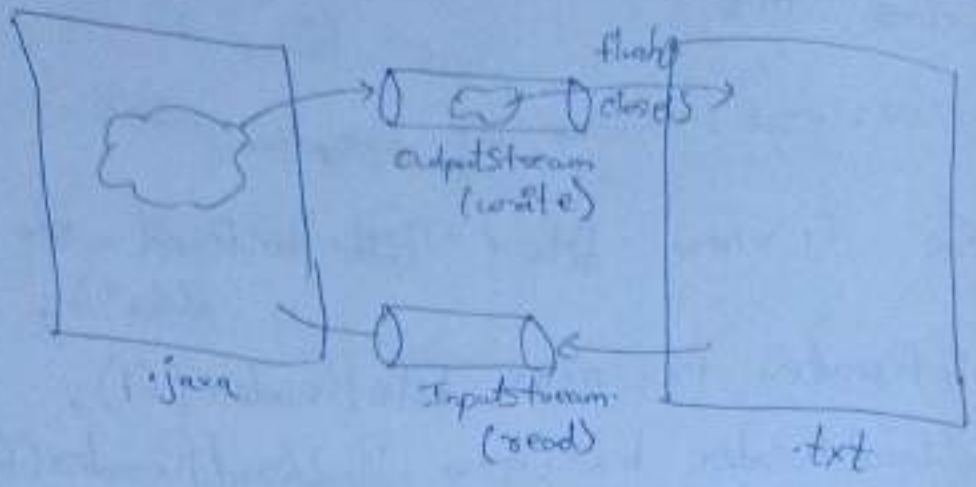
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
```

```
public class MCS
{
    psvm(SES args) throws IO Exception
    {
        File f1 = new File("Path to Read a file line by
                                line");
        FileReader fr = new FileReader(f1);
        BufferedReader br = new BufferedReader(fr);
        String line = br.readLine();
        while (line != null)
        {
            syso(line);
            line = br.readLine();
        }
    }
}
```

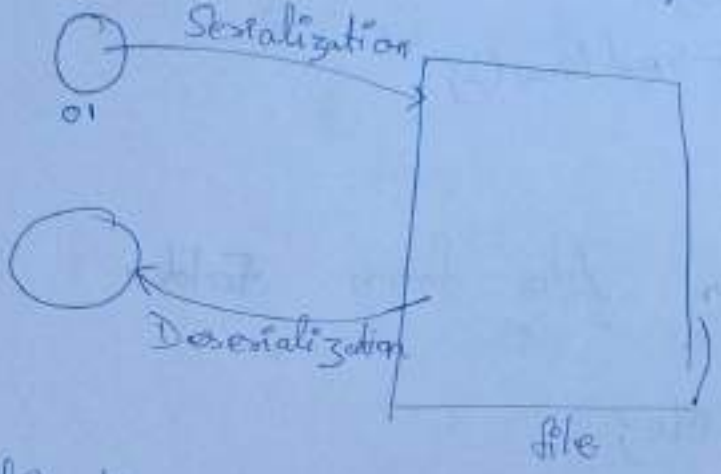
To get particular files from Folder

eg:-

```
import java.io.File;
P C M6 {
    psvm (String[] args)
    {
        File f1 = new File ("select folder path");
                                to Search
                                particular files.
        String[] names = f1.list();
        for ( String name : names)
        {
            if (name.endswith(".java"))
                Syso(name);
        }
    }
}
```



Serialization will be done only by the type of Serializable. (marker Interface)



Serialization

```
import java.io.Serializable;
public class Employee implements Serializable
{
    int id;
    String name;
    transient double salary;
    public Employee(int id, String name, double salary)
    {
        super();
        this.id=id; this.name=name; this.salary=salary;
    }
```

```
public String toString() {
    return "Employee [id=" + id + ", name=" + name + ", salary=" +
            salary + "]";
    }
}
```

To object change to file

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
public class MC7
{
    psvm (String[] args) {
    Employee emp1 = new Employee (123, "ABC", 5000);
    File f1 = new File ("employee.ser");
    try
    {
     FileOutputStream fos = new FileOutputStream (f1);
     ObjectOutputStream oos = new ObjectOutputStream
                                    (fos);
     oos.writeObject (emp1);
     oos.flush();
     fos.flush();
     oos.close();
     fos.close();
     Syso ("Serialization is done...");
    } catch (IOException e) { e.printStackTrace(); }
}
}
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
public class mc8
{
PSVM (String[] args)
 {
  File f1 = new File("employee.ser");
  try
  {
  FileInputStream fis = new FileInputStream(f1);
  ObjectInputStream ois = new ObjectInputStream(
  Employee emp1 = (Employee) ois.readObject();
  SysO(emp1);
 }
 Catch (IOException | classNotfoundException e)
  {
   e.printStackTrace();
  }
 }
}
```

Dead Lock

```
class Tickets
{ int availableTickets = 2;
  Synchronized void bookTicket()
    {
    String name = Thread.currentThread().getName();
    Syso(name + " is trying to book ticket ... ");
    if (availableTickets > 0)
      {
       Syso(name + " is got the ticket ... ");
       availableTickets--;
       Syso("Remaining Tickets = " + availableTickets);
      }
    else
    { syso(name + "is not got the ticket going
      wait by releasing it");
      try {
        wait(); }
      catch (InterruptedException e)
        {
          e.printStackTrace();
        }
      bookTicket();
    }
}
Synchronized Void CancelTicket()
    String name = Thread.currentThread().getName();
    Syso(name + " is Cancelling tickets ... ");
    availableTickets++;
```

```
        notifyAll();
    }
}
class BookTicketThread extends Thread
    Tickets ticket;
    BookTicketThread(Tickets ticket)
        {
          this.ticket = ticket;
        }
    public void run()
        {
          ticket.bookTicket();
    } }
class CancelTicketThread extends Thread
    {
      Tickets Ticket;
      CancelTicketThread(Tickets ticket)
        {
          this.ticket = ticket;
        }
      public void run()
        {
          ticket.cancelTicket();
        }

public class MC {
    PSVM(S[] args) throws InterruptedException
    {
        Tickets ticket = new Tickets();
    BookTicketThread t1 = new BookTicketThread(ticket);
                      t2 =
                      t3 =
                      t4 =
                      t5 =
```

```
CancelTicketThread t6 = new CancelTicketThread(ticket);
                t7 =                    "
t1.setName("t1");
t2.    "    ("t2");
t3.    "    ("t3");
t4.    "    ("t4");
t5.    "    ("t5");
t6.    "    ("t6");
t7.    "    ("t7");

t1.start();
t2.start();
t3.start();
t4.start();
t5.start();
Thread.sleep(500);
t6.start();
t7.start();
    }
}
```

Digital Clock

```
public class clock{
    public static void main(String args[]) {
        for(int h=0; h<24; h++) {
            for(int m=0; m<60; m++) {
                for(int s=0; s<60; s++) {
                System.out.printf("%02d:%02d:%02d\n",
                                  h,m,s);
                try { Thread.sleep(1); }
                catch(InterruptedException e) {
                    e.printStackTrace(); } } } } }
```

File Handling => Implies how to read from and write to file in java

=> Java Provides the Basic I/O package for reading and writing Streams.

=> java.io package allows to do all Input and Output task in java

What is Stream

> Sequence of Data

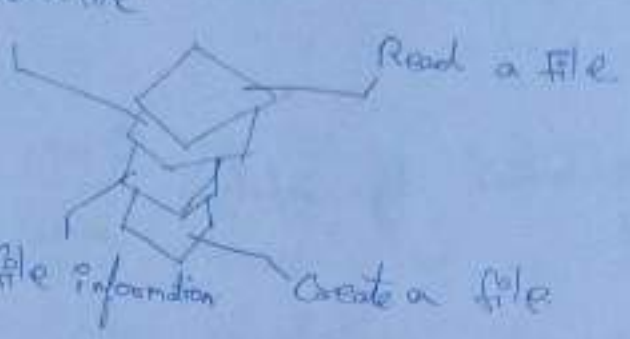
> types < Byte Stream (Incorporates with Byte Data)
Character Stream (Incorporates with Character Data)

File Methods to Perform Some Operations

CanRead() => Tests whether file is readable or not
CanWrite() => Test whether the file is writeable or not
CreateNewFile() => Creates an empty file.
Delete() => Deletes a file
Exists() => Test whether the file exists or not
getName => Returns the name of file
getAbsolutePath() => Returns the absolute pathname of the file
Length() => Returns the size of the file in bytes
List() => Returns the array of file in the directory
Mkdirs() => Creates a directory.

File

File Operations in Java



Serialization and Deserialization

1) The process of writing the object into a file is known as Serialization.
2) The process of Reading the object from the file is known as Deserialization
3) We can Serialize an object if it is of type Serializable
4) Serializable is a Marker Interface present in java.io. package
5) The following Members we can't Serialize
   i) Static Members
   ii) Transient Member
6) Serialization is only for Data Members not for Functional Members

Java types

- > Class
- > Interface
- > Enum (to achive typesafety)
- > Annotations
- > Records

Enum (Enumeration) (Java.time Package)

=> Enum will written outside class and inside package declaration

=> by default Enum methods in Enum are final and static

=> we can use Built in Enum

eg:-

```
import java.time.DayOfWeek;
enum Days
    {
      MON, TUE, WED, THUR, FRI, SAT, SUN
    }
public class MC{
        PSVM(String[] args )
            {
              Syso("Today is"+Days.TUE);
              Syso(DayOfWeek.TUESDAY);
            }
}
```

Annotation [@]

- Used to pass Some Information on Runtime Execution
- If You want to carry the Information through the Compilation process we are Annotation
- we can use annotation any where in the program

Ex:

```
@interface MyAnnotation
{

}
class A {
    void test() {
        System.out.println("Testing");
    }
class B extends A
{ @override
  void test() {
      System.out("Tested");
  }
```

Records (Similar to Java Beans but not Implicit)

→ works as a Constructor

→ ~~If is~~ we can't change the Data work like Immutable because no setter methods

→ The object is not a class object it is Record object

→ No need of getters and setters.

eg:-

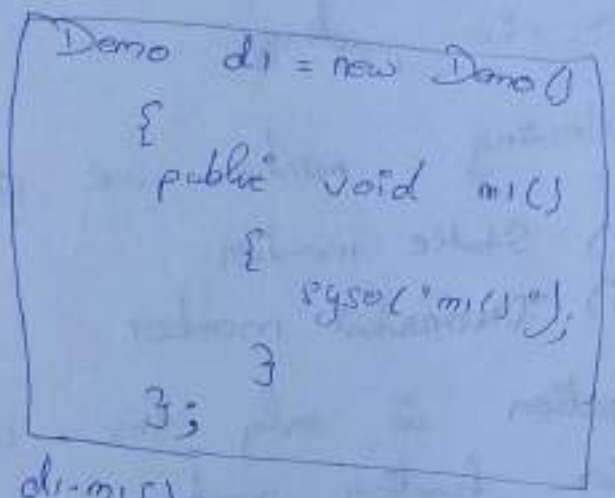
```
record Employee (int id, String name, double sal)
{

}
Public class MC{
        PSVM (S args[]) {
            // Record object
            Employee em = new Employee (123, "AB", 20
            Syso (em.id()); // Static methods
            Syso(em.name());          created
            Syso(em.sal());
        }
}
```

⇒ Data Members are Private and final

Annonamous Class

Implementing the Abstract method inside the main class

eg:- @ functionalInterface
interface Demo
{ void m1();
}
public class MC
{
psvm(String[] args)
{



Annonamous Class

d1.m1();
}
}

o/p
m1()

Demo d2 = (b) -> //Lambda Expression. only for functional Interface only one abstract method can override)
{ sysout("m1()");
};
d2.m1(10);

Serialization & Deserialization:-

⇒ The process of writting the object into a file is known as Serialization.

⇒ The process of reading the object from the file is known as deserialization

⇒ We can Serialize an object if it is of type Serializable.

⇒ Serializable is a marker Interface Present in java.io. package.

⇒ The following member we can't Serialize

i) Static member

ii) Transient member

⇒ Serialization is only for datamembers not for function members