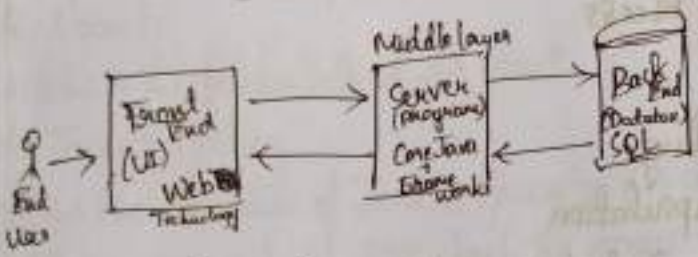
Java Full Stack Developer

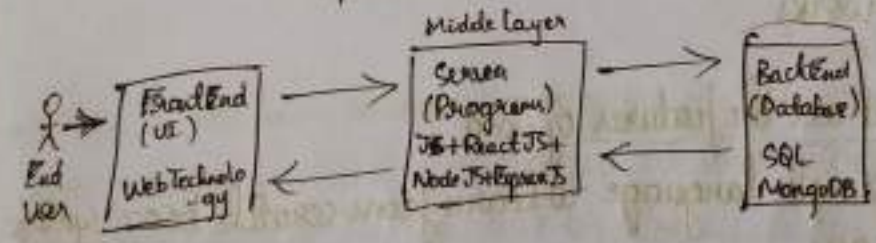
1) Core Java
2) SQL
3) Web Technologies
   - HTML
   - CSS
   - Java Script
4) Frameworks
   - Hibernate
   - Spring

Software Application



MERN Stack Developer

1) Java Script
2) SQL → MongoDB
3) Web Technology
   - HTML
   - CSS
   - Java Script
4) Frameworks
   - Node JS
   - Express JS



Java

Java is a high level programming language.

Programming language is a medium used to communicate with the system or a machine.

I) What are the course content in java?

* Variables
* Data types
* Operators
* Structure
* Looping statements
* Conditional statements
* Functions

II)

1. Object, class, keywords and identifiers
2. Methods, constructors and Blocks
3. Inheritence
4. Overloading and Overriding
5. Access Modifiers and Encapsulation
6. Casting, Abstract Keyword, Interface and Arrays
7. Polymorphism
8. Abstraction

III)

1) Java libraries → Strings
2) Exception handling
3) File handling
4) Multi-Threading
5) Collection Frameworks

* Why Java or What are the features of Java?

① Java is a very simple language to learn from scratch bcoz syntax in java is very easy.

② Java is platform independent. which means java program can be executed in windows, linux, unix, mac... etc OS

③ Java is object oriented language.
Object means which appears physically.
Ex:- Pen, chair etc,..

④ Java is extensible, means java program we can integrate with C++ or python etc to develop a application/product/project more effectively.

⑤ Java is robust, means it can handle huge application and is very strong. No virus will harm it.

⑥ Java has a automatic garbage collector which identifies the duplicate data and automatically deletes it.

* Which all are the companies make use of java?
: Starting from startup company to top MNC company make use of java

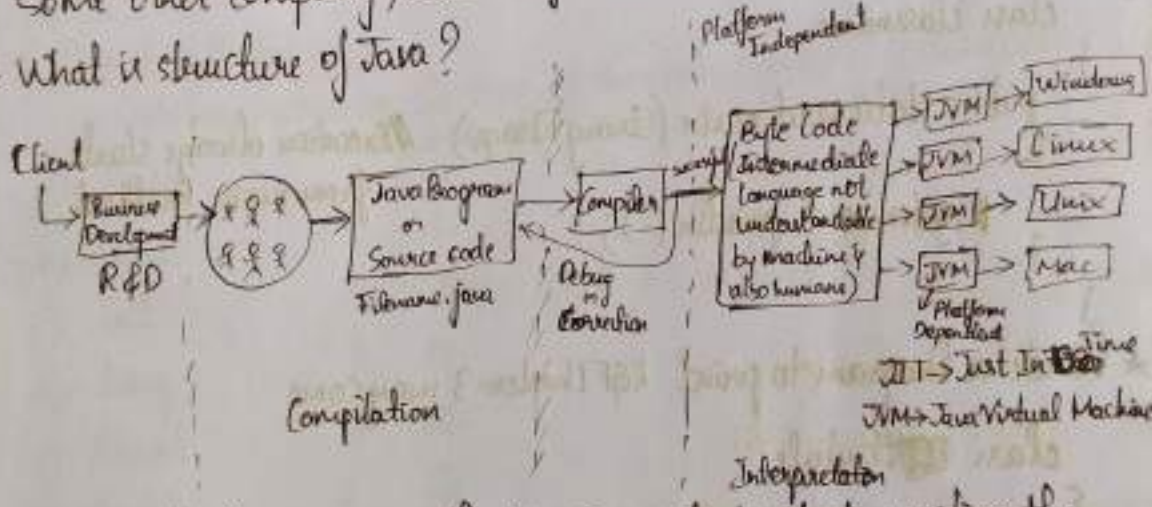
* Which company introduced java?
A: Sun MicroSystems (S.M.S)

Who is the owner of java?
A: Oracle.

* Who is the person introduced java?
A: James Gosling

The First name of Java is Green talk but it was not so popular, so they renamed as OAK, but the name was already registered by some other company, later they renamed as Java (Coffee Bean)

* What is structure of Java?



→ When a developer recieves the requirement, he starts writing the java program which is also known as Source Code, Once he completes the java program, he will save as filename.java.

→ To check any errors (mistakes/bugs) the developer will give to compiler for compilation.

→ Compiler: compiler will check any bugs in a program, if it finds any bugs it will send back to the developer to debug (correction) by throwing a error. Developer will make correction and gives to compiler again.

→ Once compiler finds no bug in the program, the compiler will convert java program to byte code and save file as filename.class

→ Byte Code :- Byte code is an intermediate code which is non understand-able by machines nor human being.
Byte code is platform independent we can run in windows, linux, unix and others OS with the help of JVM (Java Virtual Machine)

→ JVM :- JVM reads the byte code & convert to machine understand-able language with the help of JIT (Just In Time) Compiler and JVM is platform dependant means JVM for windows we can't use for linux or any other OS.

→ The first half is compilation & the second half is interpretation.

★ Structure of Java Program?

1) Class
2) Main method
3) print statement

```
class classname
{
  public static void main (String[]args)   //Execution always starts from main method
  {
    System.out.println( );
  }
}
```

★ Write a Program to print KGF Chapter-3 may come

```
class thambale
{
  public static void main(String[]args)
  {
    System.out.println("KGF Chapter-3 may come");
  }
}
```

//→ Single line comment
/*...*/→ Multi line comment
\n → new line
\t → tab space

Search

↳ Notepad → Write Program → ~~Save as filename.~~ Save filename.java → Desktop → Create a folde

→ save as filename.java & All files

For execution:-

Go to folder & type cmd

For compilation:-

javac filename.java (To check errors)

For Interpretation:- (For O/P)

java filename

Variables:- Variables are a container to store the information or data

Ex:- ① age = 21 [21] age

② height = 5.2 [5.2] height

Types:-

1) Primitive (Datatypes):-

Datatypes indicates the type of the data stored in a variable

	Byte	bits
a) Byte	1	8
b) Short	2	16
c) int	4	32
d) long	8	64
e) float	4	32
f) double	8	64
g) char	2	16 / 'A'
h) boolean	1 bit either 0/1 or T/F	

2) Non Primitive:

class

enum

Interface

Variable Declaration:-

Syntax:- Type variablename

Ex: ① int age

② double height

Variable Initialization:-

Syntax: variablename = value

Ex ① age = 21

② height = 5.2

Variable Declaration & Initialization:-

Syntax:- Type variable name = value

Ex: ① int age = 21

② double height = 5.2

③ String name = "dinga"

④ char gender = 'M'

1) Write a program to display person age & height

```
class Person
{
    public static void main (String[] args)
    {
        int age = 21;
        double height = 5.2;
        System.out.println (age + " years");
        System.out.println (height + " feet");
    }
}
```

Concatenation

O/P:-

21 years

5.2 feet

Assignment:-

~~Combine both print lines~~ WAP height, age & gender of tom & jerry

```
class TomJerry
{
    public static void main(String[] args)
    {
        int age = 21;
        double height = 5.5;
        char gender = 'M';
        int age1 = 9;
        double height1 = 4.5;
        char gender1 = 'F';
        System.out.println("Tom age is "+age+" years and height is "+height
        +" feet with gender "+gender);
        System.out.println("Jerry age is "+age1+" years and height is
        "+height1+" feet with gender "+gender1);
    }
}
```

O/p:- Tom age is 21 years and height is 5.5-feet with gender M
Jerry age is 9 years and height is 4.5-feet with gender F

Diff b/w print & println

print :- Print the set of statements and control with in same line

println :- —— || —— control goes to next line (\n)

Operators:- Operators are the symbols used to perform some task

1) Arithmetic Operators (+, -, *, /, %)

```
class AriOper
{
    public static void main(String[] args)
    {
        int a = 20;
        int b = 10;
        System.out.println(a+b);
        System.out.println(a-b);
        System.out.println(a*4);
        System.out.println(a/b);
        System.out.println(b%3);
    }
}
```

O/p:-
30
10
80
2
1

2) Assignment Operators :- (+=, -=, *=, /=, %=)

```
class AssignmentOperators
{
    public static void main (String[] args)
    {
        int a=10;
        int b=5;
        int c=4;
        int d=8;
        int e=5;
        System.out.println(a+=5);
        System.out.println(b-=10);
        System.out.println(c*=3);
        System.out.println(d/=3);
        System.out.println(e%=21);
    }
}
```

O/P :-

15
-5
12
2
5

```
class AssignmentOperators
{ public static void main(String[] args)
  {
    int a=10;
    System.out.println(a+=5);
    System.out.println(a-=15);
    System.out.println(a*=3);
    System.out.println(a/=2);
    System.out.println(a%=3);
  }
}
```

O/P :-

15
0
0
0
0

3) Relational/Comparison Operators :- (>, <, >=, <=, ==, !=) Gives boolean Result

```
class RelationalOperators
{
    public static void main (String[] args)
    {
        int x=20;
        int y=10;
        SOP(x>=y);
        SOP (x==15);
```

```
SOP(y>=15);
SOP(x<=30);
SOP(y==10);
SOP(x!=y);
```

O/P:
true
false
false
true
true
true

```
    }
}
```

4) Logical Operators:- (||, &&, !) Gives boolean result

Real Time Ex:- OR

Mobile → Pattern
+
PIN
+
Fingerprint
+
Face ID

Ex:

Gmail → username
+
→ password

```
class LogicalOperators
{
    public static void main(String[] args)
    {
        int x=20;
        int y=10;
        boolean res = (x>=y && x==10);
        SOP(res);
        SOP(x>=y || x==10)
        SOP(x==y || y==10)
        SOP(x>y && y==10)
        SOP(!true)
    }
}
```

O/P:-
false
true
true
true
false

5) Ternary Operators:-

Syntax:- Condition ? T:F

Ex:- res = a > b ? a : b
100 20 100 20

```
class TernaryOperators
{
    public static void main(String[] args)
    {
        int a=10;
        int b=20;
        int max = a>b?a:b;
```

```
        SOP("Maximum of "+a+" and "+b+" is "+max);
    }
}
```

O/p:-

Maximum of 20 and 20 is 20

Assignment question:

WAP to find minimum number of p & q (80, 40) (10, 50)

```
Class TernaryOperator
{
    public static void main(String[] args)
    {
        int p=80; //p=10
        int q=40; //q=50
        int min = a<b?a:b;
        SOP("Minimum of "+p+" and "+q+" is "+min);
    }
}
```

O/p:-

Minimum of 80 and 40 is 40

Minimum of 10 and 50 is 10

6) Unary Operators:-

Increment	Decrement
++ => Increase the value by 1	-- => Decrease the value by 1

Post Increment	Pre Increment	Post Decrement	Pre Decrement
x = 1 (x: 2) y = x++ (y: 1) First assign then increase value by 1	a = 1 (a: 2) b = ++a (b: 2) First increase value by 1 then assign	p = 1 (p: 0) q = p-- (q: 1) First assign then decrease value by 1	i = 1 (i: 0) j = --i (j: 0) First decrease value by 1 then assign

```
Class UnaryOperator
{
    public static void main(String[] args)
    {
        int x = 1;
        int y = x++;
        SOP("x value is:" +x+ "y value is:"+y);
```

```
int a = 1;
int b = ++a;
SOP ("a value is: "+a+" b value is: "+b);
int p = 1;
int q = p--;
SOP ("p value is: "+p+" q value is: "+q);
int i = 1;
int j = --i;
SOP ("i value is: "+i+" j value is: "+j);
}
}
```

O/P:-

x value is : 2 y value is : 1
a value is : 2 b value is : 2
p value is : 0 q value is : 1
i value is : 0 j value is : 0

Ex:-

1) $X = 1$
$Y = \underset{1}{X^{++}} + \underset{2}{X^{++}}$
$= 1 + 2$

$\boxed{123}$ X
$\boxed{3}$ Y

2) $X = 1$
$Y = \underset{2}{++X} + \underset{3}{++X}$
$= 2 + 3$

$\boxed{123}$ X
$\boxed{5}$ Y

3) $X = 1$
$Y = \underset{1}{X^{--}} + \underset{0}{X^{--}}$
$= 1 + 0$

$\boxed{1\,0\,-1}$ X
$\boxed{1}$ Y

4) $X = 1$
$Y = \underset{0}{--X} + \underset{-1}{--X}$
$= 0 + (-1)$
$= 0 - 1$

$\boxed{0\,-1}$ X
$\boxed{-1}$ Y

```
class UnaryOperator
{
    public static void main(String[] args)
    {
        int x=1;
        int y= x++ + x++;
        SOP("x:"+x+" y:"+y);
        int a=1;
        int b=++a+++a;
        S.O.P("a:"+a+" b:"+b);
        int p=1;
        int q= p--+p--;
        S.O.P("p:"+p+" q:"+q);
        int i=1;
        int j=--i+--i;
        S.O.P("i:"+i+" j:"+j);
    }
}
```

O/P:-

x: 3 y: 3

a: 3 b: 5

p: -1 q: 1

i: -1 j: -1

Ex:-

1) x = 3

Y = x++ - x-- + --x + ++x

3 - 4 + 2 + 3

-1 + 5

= 4

x	y
3 4 2 1 3	4

2) x = 0

Y = x-- - x++ - --x - x--

0 - 1 - -1 - -1

=

= 0+1+1+1

= 3

x	y
0 -1 0 -1 -2	3

Assignment Question

3) $X = 5$

$Y = X^{++} + --X - X^{--} + ++X + X^{--} - --X + X^{--} + --X$

[redacted]

4) $X = 0$

$Y = X^{++}$

$Y = X^{++}$

$Y = X^{++}$

$\boxed{X = 3 \quad Y = 2}$

$\boxed{0123}$ X $\boxed{012}$ Y

5) $X = 0$

$X = X^{++}$

$X = X^{++}$

$X = X^{++}$

$\boxed{0}$ X [redacted]

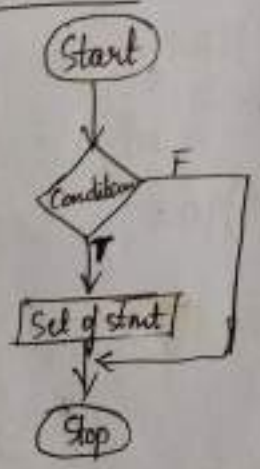
Conditional Statements:-

1) Simple if
2) if else
3) if else if
4) Nested if
5) Switch

Simple if (+ve scenario)

Syntax:- if (condition)
{
set of stmt
}

Flow chart:-



WAP to check no is ≥ 10.

```
class SimpleIf
{
    public static void main (String[] args)
    {
        int n = 15;
        if(n >= 10)
        {
            SOP(n+" is greater than equal to 10");
        }
    }
}
```

O/P:-

Ex① n=15

15 is greater than or equal to 10

② n=10

10 is greater than or equal to 10

③ n=5

∴ There will be no output

WAP to check the person is eligible to vote

```
/*class [redacted] SimpleIf1
{
    public static void main (String[] args)
    {
        int n;
        [redacted] Scanner sc = new Scanner(system.in);
        if(n≥18)
        {
            SOP("The person is eligible to vote");
        }
        SOP("Not eligible");
    }
}*/
class SimpleIf2
{
    public static void main (String[] args)
    {
        int age=21;
```

```
        if(age >= 18)
        {
            S.OP("Person is eligible to vote, pls vote");
        }
    }
}
```

O/P:-

① age = 21

Person is eligible to vote, pls vote

② age = 17

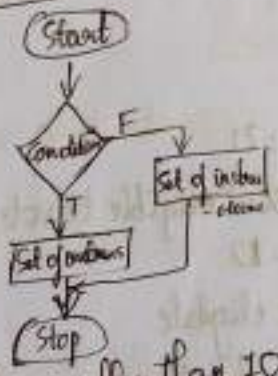
∴ No output

→ **if else :-** (+ve scenario & -ve scenario)

Syntax:-

```
if(condition)
{
    set of instructions
}
else
{
    set of instructions
}
```

Flow Chart :-



WAP to check the number is greater than or equal or smaller than 10

```
class IfElse
{
    public static void main(String[] args)
    {
        int n=15;
        if(n>=10)
        {
            S.O.P("n is >= 10")
        }
        else
        {
            S.O.P("n is < 10")
        }
    }
}
```

O/P:-

① n = 15

15 is >= 10

② n = 5

5 is < 10

WAP to check if the person is eligible to vote

```
class IfElse1
{
    public static void main(String[] args)
    {
        int age = 21;
        if (age >= 18)
        {
            S.O.P("Person is eligible to vote, pls vote");
        }
        else
        {
            S.O.P("Not eligible");
        }
    }
}
```

Op:-

① age = 21
Person is eligible to vote, pls vote

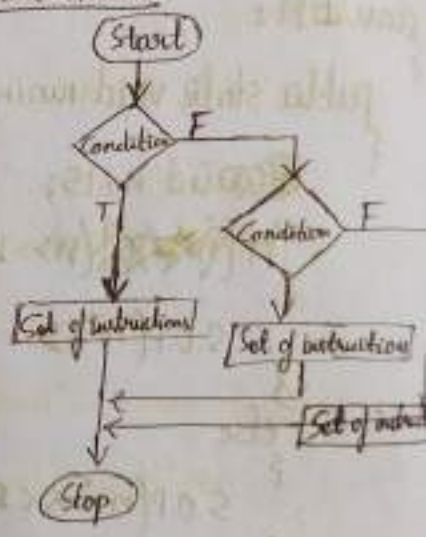
② age = 17
Not eligible

→ if else if :-

Syntax:-

```
if (condition)
{
    Set of instructions
}
else if (condition)
{
    Set of instructions
}
else
{
    set of instructions
}
```

Flowchart:-



WAP to compare 2 numbers

```
class IfElseif
{
    public static void main(String[] args)
    {
        int a = 20;
        int b = 10;
```

```
        if(a>b)
        {
            S.O.P(a+" is greater than "+b);
        }
        else if(a==b)
        {
            S.O.P(a+" is equal to "+b);
        }
        else
        {
            S.O.P(a+" is smaller than "+b);
        }
    }
}
```

O/P:-

1. a=20
   b=10
   20 is greater than 10
2. a=10
   b=10
   10 is equal to 10
3. a=2
   b=10
   2 is smaller than 10

WAP to check gender of a person

```
class IfElseIf1
{
    public static void main(String[] args)
    {
        String gender="Male"; or char gender='M'
        if(gender=="Male") // if(gender=='M')
        {
            S.O.P("Person is Male");
        }
        else if(gender=="Female") // else if(gender=='F')
        {
            S.O.P("Person is Female");
        }
        else
        {
            S.O.P("Person is Transgender");
        }
    }
}
```

O/P:-

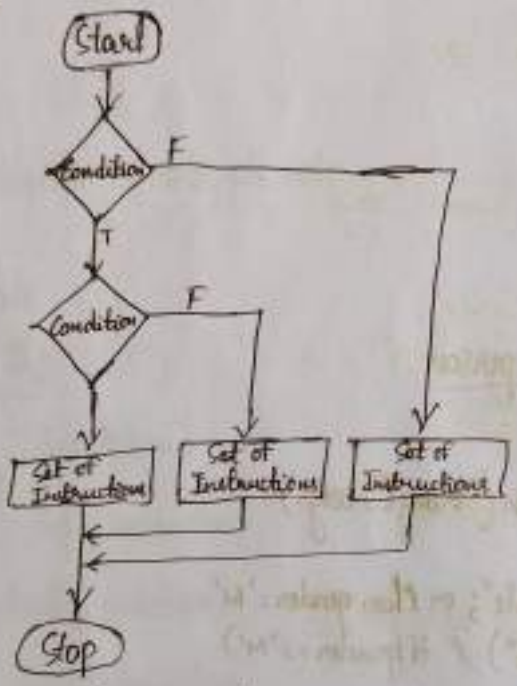
1) gender='M' O/P:- Person is Male
2) gender='F' O/P:- Person is Female
3) gender='H' O/P:- Person is Transgender
4) gender='T' O/P:- Person is transgender

→ Nested if :-

Syntax:-

```
if(condition)
{
    if(condition)
    {
        Set of instructions
    }
    else
    {
        Set of instructions
    }
}
else
{
    Set of instructions
}
```

Flow Chart:-



WAP to compare two numbers

```
class NestedIf
{
    public static void main(String[] args)
    {
        int a=20;
        int b=10;
        if(a>=10)
        {
            if(a>b)
            {
                S.O.P("a" is greater than "b");
            }
            else
```

```
        {
            S.O.P("a+" equal to "+b);
        }
    }
    else
    {
        S.O.P(a+" is smaller than "+b);
    }
}
}
```

O/P:-

① a=20, b=10 O/P:- 20 is greater than 10

② a=10, b=10 O/P:- 10 is equal to 10

③ a=5, b=10 O/P:- 5 is smaller than 10

WAP to check for gmail/email login

```
class NestedIf1
{
    public static void main (String[] args)
    {
        String username = "john";
        String password = "john403";
        if(username == "john")
        {
            if(password == "john403")
            {
                S.O.P("Login Successful");
            }
            else
            {
                S.O.P("Invalid password");
            }
        }
        else
        {
            S.O.P("Invalid username");  "Login Unsuccessful"
        }
    }
}
```

O/P:-

① username = "john", password = "john403" O/P:- Login Successful

② username = "john" O/p :- Invalid Password
password = "JOHN403"

③ username = "John" O/p :- Invalid Username / Login Unsuccessful
password = "john403"

→ Switch :-

Syntax :- switch (choice)
{
case 1 : set of instructions
break;
case 2 : set of instructions
break;
case 3 : set of instructions
break;
:
case n : set of instructions
}

```
class Switch
{
    public static void main (String[] args)
    {
        int choice = 1;
        switch (choice)
        {
            case 1 : SOP("in case 1");
            break;
            case 2 : SOP("in case 2");
            break;
            case 3 : SOP("in case 3");
            break;
            default : SOP("invalid choice");
        }
    }
}
```

WAP to display student overall

```
class Switch1
{
    public static void main (String[] args)
    {
        char result = 'A';
        switch (result)
        {
            case 'A' : SOP ("Excellent");
            break;
```

```
        case 'B': SOP("Good");
        break;
        case 'C': SOP("Average");
        break;
        case 'D': SOP("Fail");
        break;
        default: SOP("Invalid result");
    }
  }
}
```

O/p:
1) A o/p:- Excellent
2) B o/p:- Good
3) C o/p:- Average
4) D o/p:- Fail
5) F o/p:- Invalid Result

Assignment Question:-

1. WAP to check greatest no among a, b, c
2. WAP to check student result
   80-100 : Distinction
   60-80 : First Class
   35-60 : Pass
   0-35 : Fail
3. WAP to check eligible for marriage in matrimony (M-21, F-18)

Looping Statements:- Whenever we need to repeat the task, we use looping statements.

1) For loop
2) While loop
3) Do while loop
4) Nested for loop

→ For loop:- whenever we know starting point & ending point, we use for loop

Syntax:-
```
for (Initialization; Condition; Updation)
{
    Set of statements;
}
```

Ex: Print 'hello' 5 times

```
class Hello
{
  public static void main(String[] args)
  {
    for (int i=1; i<=5; i++)
    {
      S.O.P("Hello");
    }
  }
}
```

O/p:- Hello
Hello
Hello
Hello
Hello

Programs:

1) WAP to print numbers 1 to 5

```
for (int i=1; i<=5; i++)
{
    S.O.P(i);
}
```

o/p:- 1
2
3
4
5

2) WAP to print numbers 1 to 5 in descending order 5 to 1

```
for (int i=5; i>=1; i--)
{
    S.O.P(i);
}
```

o/p:- 5
4
3
2
1

3) WAP to print even numbers 1 to 10

4) ——— " ——— in reverse order

5) WAP to print odd numbers 1 to 10

6) ——— " ——— in reverse order

7) WAP to print multiple of 3 from 1 to 15

8) ——— " ——— in reverse order

3 Sol)

```
for (int i=2; i<=10; i=i+2)
{
    S.O.P(i);
}
```

o/p:- 2
4
6
8
10

4 Sol)

```
for (int i=10; i>=2; i=i-2)
{
    S.O.P(i);
}
```

o/p:- 10
8
6
4
2

5 Sol)

```
for (int i=1; i<=10; i=i+2)
{
    S.O.P(i);
}
```

o/p:- 1
3
5
7
9

6 Sol)

```
for (int i=10; i>=1; i=i-2)
{
    S.O.P(i);
}
```

o/p:- 9
7
5
3
1

7 Sol)

```
for (int i=3; i<=15; i=i+3)
{
    S.O.P(i);
}
```

o/p:- 3
6
9
12
15

(SS) for (int i=15; i>=3; i=i-3)
{
S.O.P(i);
}

O/P: 15
12
9
6
3

Assignment:- (Condition Stat)

1) WAP to check a number is positive or negative
2) ——— " ——— is even or odd
3) ——— " ——— is multiple of 4

For loop question:-

1) WAP to find sum of no 1 to 10

~~[illegible]~~

```
class ForLoop
{
    public static void main(String[] args)
    {
        int sum=0;
        for(int i=1; i<=10; i++)
        {
            sum=sum+i;
        }
        System.out.println("Sum of first 10 numbers is:"+sum);
    }
}
```

O/P:-
Sum of first 10 numbers is: 55

For Factorial:-

```
int fact=5;
int num=1;
for(int i=1; i<=fact; i++){
    num=num*i;
}S.O.P(num);
```

Assignment (1):-

1) WAP to check a no is +ve or -ve

```
class CheckPositiveOrNegative
{
    public static void main(String[] args)
    {
        int num=10;
        if(num>0)
        {
            S.O.P("No is positive");
        }
        else if(num<0)
        {
            S.O.P("No is negative");
        }
        else
        {
            S.O.P("No is zero");
        }
    }
}
```

O/P:- 1) num=10
No is positive
2) num=0
No is zero
3) num=-10
No is negative

2) WAP to check a number is even or odd

```
class EvenOdd
{
    public static void main(String[] args)
    {
        int num = 10;
        if(num % 2 == 0)
        {
            System.out.println("Number is even")
        }
        else
        {
            System.out.println("Number is odd")
        }
    }
}
```

O/P:- 1) num = 10
Number is even
2) num = 1
Number is odd

3) WAP to check a number is multiple of 4

```
class IsMultipleOfFour
{
    public static void main(String[] args)
    {
        int num = 8;
        if(num % 4 == 0)
        {
            System.out.println("It is multiple of 4");
        }
        else
        {
            System.out.println("Not a multiple of 4");
        }
    }
}
```

O/P:
1) num = 8
It is multiple of 4
2) num = 15
Not a multiple of 4

→ **While loop**:- When we know starting point & dont know ending point we use while loop.

Syntax:-
```
while(condition)
{
    set of stmt;
    updation;
}
```

Ex: ① Print numbers from 1 to 5 using while loop

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

```
int n=1;
while(n<=5)
{
    S.O.P(n);
    n++;
}
}
```

O/P:- 1
2
3
4
5

③ For reverse order

```
int n=5;
while(n>=1)
{
    S.O.P(n);
    n--;
}
```

O/P:- 5
4
3
2
1

④ even no 1 to 10

```
int n=2;
while(n<=10)
{
    S.O.P(n);
    n+2;
}
```

O/P: 2
4
6
8
10

Reverse:-

```
int n=10;
while(n>=1)
{
    S.O.P(n);
    n-2;
}
```

O/P:- 10
8
6
4
2

⑤ odd no 1 to 10

```
int n=1;
while(n<=10)
{
    S.O.P(n);
    n+2;
}
```

O/P:- 1
3
5
7
9

Reverse:-

```
int n=9;
while(n>=1)
{
    S.O.P(n);
    n-2;
}
```

O/P: 9
7
5
3
1

Assignment:-

⑥ WAP to print multiple of 3 from 1 to 22

Diff b/w for loop & while loop

For loop	While loop
* Here we know starting point and ending point.	* Here we know starting point, we don't know ending point.
* Here we know how many time to repeat the task.	* Here we don't know how many times to repeat the task.
Ex:- 5 times, 10 times	

WAP to withdraw 2000 rupees each times till it shows amount is insufficient (Realtime ex for while loop)

```
class Whileloop
{
    public static void main(String[] args)
    {
        double act_bal = 11999;
        double withdraw = 2000.00;
        while(act_bal >=2000)
        {
            S.O.P("withdraw amount:" + withdraw);
            act_bal = act_bal - withdraw;
        } S.O.P("account_balance:" + act_bal);
    }
}
```

Op:- withdraw amount : 2000.0
account_balance : 9999.0
withdraw amount : 2000.0
account_balance : 7999.0
withdraw amount : 2000.0
account_balance : 5999.0
withdraw amount : 2000.0
account_balance : 3999.0
withdraw amount : 2000.0
account_balance : 1999.0

→ Dowhile Loop:-

Syntax:-

```
do
{
    Set of instructions;
    updation;
}
while(condition);
```

Ex:- ① Print numbers from 1 to 5

```
int n=1;
do
{
```

```
    S.O.P(n);
    n++;
  }while(n<=5);
```

O/P:
1
2
3
4
5

Reverse:-

```
int n=5;
do
{
  S.O.P(n);
  n--;
}while(n>=1);
```

O/P:
5
4
3
2
1

Diff b/w while loop & Do while loop

While loop	Do While Loop
* Syntax:- `while(condition) { set of instructions; updation; }`	* Syntax:- `do { set of instructions; updation; } while(condition);`
* First checks the condition, if the condition is true executes set of instructions.	* First executes set of instructions then checks the condition.
* If condition fails it won't executes any set of instructions.	* If condition fails it executes atleast once.

→ Nested for loop:-

Syntax: `for(I;C;U)`

```
{
  for(Initialization; Condition; Updation)
  {
    set of instructions;
  }
}
```

Outer for loop

Inner for loop

Ex:-1)

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

```
        S.O.P("Program Started");
        for(int i=1; i<=3; i++)
        {
            for(int j=1; j<=4; j++)
            {
                S.O.P("i value is: "+i+" j value is: "+j);
            }
        }
        S.O.P("Program Ended");
    }
}
```

O/P:- Program Started

```
i value is: 1 j value is: 1
i value is: 1 j value is: 2
i value is: 1 j value is: 3
i value is: 1 j value is: 4
i value is: 2 j value is: 1
i value is: 2 j value is: 2
i value is: 2 j value is: 3
i value is: 2 j value is: 4
i value is: 3 j value is: 1
i value is: 3 j value is: 2
i value is: 3 j value is: 3
i value is: 3 j value is: 4
Program Ended
```

2) Print "****"

```
for(int i=1; i<=4; i++)
{
    for(int j=1; j<=4; j++)
    {
        S.O.print("*");
    }
    S.O.println();
}
```

Assignment Question: (while loop)

WAP to print multiple of 3 from 1 to 22

```
int n=3;
while (n<=22)
{
    S.O.P(n);
    n=n+3;
}
```

Assignment Questions (based on conditional statements)

1) WAP to check greatest number among a, b, c

Sol

```
class CheckGreatest
{
    public static void main(String[] args)
    {
        int a=40, b=78, c=19;
        if (a>=b && a>=c)
        {
            S.O.P(a+"is the largest number");
        }
        else if (b>=a && b>=c)
        {
            S.O.P(b+"is the largest number");
        }
        else
        {
            S.O.P(c+"is the largest number");
        }
    }
}
```

O/P:-

1) a=40, b=78, c=19
78 is the largest number

2) a=[illegible], b=40, c=19
[illegible] is the largest number

2) WAP to check student result [80-100: Distinction, 60-80: First class, 35-60: Pass, 0-35: Fail]

Sol

```
class StudentResult
{
    public static void main(String[] args)
    {
        int z=95;
        if(z>=80 && z<100)
        {
            S.O.P("Excellent");
        }
        else if (z>=60 && z<80)
        {
            S.O.P("Good");
```

```
else if (z>=35 && z<60)
{
    S.O.P("Average");
}
else if (z<35)
{
    S.O.P("Fail");
}
else
{
    S.O.P("Not valid");
}
}
}
```

O/P:
1) z = 95
Excellent
2) z = 78
Good
3) z = 39
Average
4) z = 110
Not valid

4) WAP to check eligibility for marriage in matrimony (M-21, F-18)

Sol

```
class Marry
{
    PSVM()
    {
        int boyage = 22;
        int girlage = 19;
        ~~if (boyage >= 21 && girlage >= 18)~~
        ~~{~~
            ~~S.O.P("Eligible for marriage");~~
        ~~}~~
        ~~else if~~
        ~~{~~
            ~~S.O.P("Not eligible");~~
        ~~}~~
        ~~else if (boyage >= 21)~~
        ~~{~~
            ~~S.O.P("Boy can marry");~~
        ~~}~~
        ~~else if (girlage >= 18)~~
        ~~{~~
            ~~S.O.P("girl can marry");~~
        ~~}~~
        ~~else~~
        ~~{~~
        if (boyage >= 21 && girlage >= 18)
        {
            S.O.P("Both are eligible");
        }
```

O/P:-
1) boyage = 21
girlage = 18
Both are eligible
2) boyage = 20
girlage = 17
Both are not eligible
3) boyage = 21
girlage = 17
Boy can marry and girl can't marry
4) boyage = 20
girlage = 19
Girl can marry and boy can't marry

```
        else if (boyage >= 21 && girlage <= 18)
        {
            S.O.P("Boy can marry and girl can't marry");
        }
        else if (boyage <= 21 && girlage >= 18)
        {
            S.O.P("Girl can marry and Boy can't marry");
        }
        else
        {
            S.O.P("Both are not eligible");
        }
    }
}
```

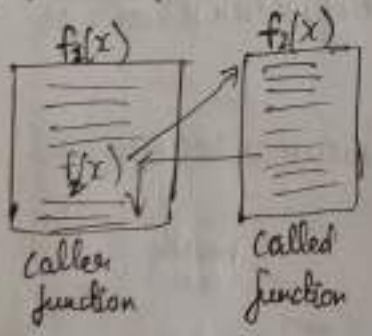
Functions: → Function is a block of codes written for reusability and modularity.

* Function is also known as "methods" technically

Syntax:-

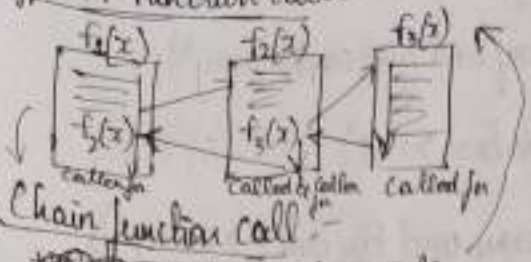
```
                                                                  variables
                  public                static                void
                                                                 int
Access specifier modifier return type function_name (parameter)
                   1                    2          3          4          5
{
    ____
    ____
    ____
    return;
        6
}
```

1, 2, 5 are optional. 3 & 4 are must



* The function which calls other function is called caller function.
* The function which gets called is called called function.
* Whenever we need to call the function we need to call by its name for execution.
* The functions we can write before the main or after the main method in java.

Types of function call:-

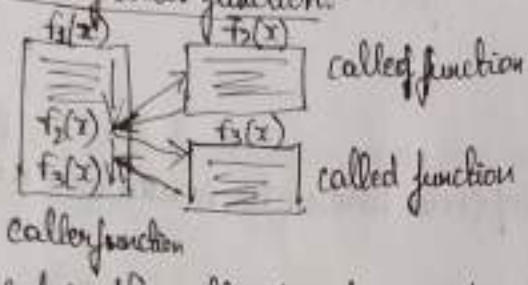


① Chain function call:-

~~Sequence of function call~~

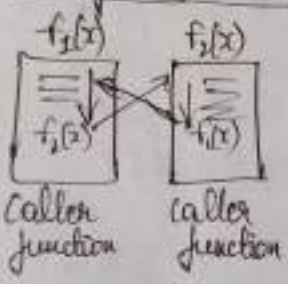
* Here f_1 is the caller function to f_2 & f_2 is the caller function to f_3

② Sequence of calls function:-



* Here f_1 is the caller function to f_2 & f_3

③ Recursive function call:-



* When called function calls back the caller function is known as recursive call function
* When caller function calls itself is also known as recursive call function.

Function Parameter:-

Syntax:- Func_name (parameter type1, parameter name1, parameter type 2, parameter name2

Variable name

Datatype

* Caller function passes the value to called function with the help of parameters is called passing value.
* Called function returns back the value to the caller function is called returning value.
* The caller function can recieve the value or it can ignore it.

Passing	Returning

Programs:

1) Adding 2 numbers (without parameters & return type)

```
class Functions
{
    public static void add()
    {
        int a=10;
        int b=20;
        int c=a+b;
        S.O.P(c);
    }
    public static void main(String[] args)
    {
        add();
    }
}
```

O/P:-
30

2) WAP to find square of a number

```
class Function1
{
    public static void sqr()
    {
        int a=15;
        int b=a*a;
        S.O.P(b);
    }
    public static void main(String[] args)
    {
        sqr();
    }
}
```

O/P:-
225

3) WAP to find both adding of 2 numbers & square of a number

```
class Function
{
    public static void add()
    {
        int a=10;
        int b=20;
        int c=a+b;
        S.O.P(c);
```

```
    public static void sqr()
    {
        int a = 15;
        int b = a*a;
        S.O.P(b);
    }
    P S V m (String[] args)
    {
        add();
        sqr();
    }
}
```

O/P:-
30
225

With parameter & without returntype

1) Add 2 numbers

```
class Function
{
    public static void add(int a, int b)
    {
        int c = a+b;
        S.O.P(c);
    }
    psvm()
    {
        add(10,20);
    }
}
```

O/P:-
30

2) Square of number

```
class Function
{
    public static void sqr (int a)
    {
        int b = a*a;
        S.O.P(b);
    }
    psvm()
    {
        sqr(5);
    }
}
```

O/P:-
25

Assignment:-

1) WAP to multiply 2 numbers (2)

2) -//- to find simple interest (3)

Without parameter & without return type

1) Sol 1)

```
class Functions
{
    public static void mul()
    {
        int a = 15;
        int b = 9;
        int c = a*b;
        S.O.P(c);
    }
    psvm()
    {
        mul();
    }
}
```

O/P:-
135

with parameter & without return type

2)

```
class Functions
{
    public static void mul(int a, int b)
    {
        int c = a*b;
        S.O.P(c);
    }
    psvm()
    {
        mul(15,9);
    }
}
```

O/P:-
135

Without parameter & without return type

2) Sol 1)

```
class Functions
{
    public static void si()
    {
        double p = 13000;
        double t = 2.5;
        double r = 12;
        double si = (p*r*t)/100;
        S.O.P(si);
    }
    psvm (String[] args)
    {
        si();
    }
}
```

O/P:-
3900.0

parameter and without returntype

4) class Functions
{
public static void si(~~int a, int b~~)(double p, double t, double r)
{
double si = (p*r*t)/100;
S.O.P(si);
}
psvm(String[] args)
{
si(13000, 2.5, 12);
}
}

O/P:-
3900.0

5) class Functions
{
public static void mul ()
{
int a = 15;
int b = 9;
int c = a*b;
S.O.P(c);
}
public static void si()
{
double p = 13000;
double t = 2.5;
double r = 12;
double si = (p*r*t)/100;
S.O.P(si);
}
psvm(String[] args)
{
mul();
si ();
}
}

O/P:-
135
3900.0

6) class Functions
{
public static void mul()
{
int a = 15;
int b = 9;
int c = a*b;

```
        si();
        s.o.p(c);
    }
    public static void si()
    {
        double p=13000;
        double t=2.5;
        double r=12;
        double si=(p*r*t)/100;
        s.o.p(si);
    }
    p s v m(String[] args)
    {
        mul();
    }
}
```

O/P:-
3900.0
135

Without parameter and with returntype

1) WAP to print product of 2 numbers

```
class Product
{
    public static int pro()
    {
        int a=5;
        int b=6;
        int c=a*b;
        return c;
    }
    public static void main(String[] args)
    {
        s.o.p(pro());
    }
}
```

O/P:-
30

2) WAP to print simple interest

```
class SimpleInterest
{
    public static double si()
    {
        double p=13000;
        double t=2.5;
        double r=12;
        double si=(p*r*t)/100;
```

```
        return si;
    }
    p s v m(String[] args)
    {
        S.O.P(si());
    }
}
```

O/P:
3900.0

Sequence call

```
public static int pro()
{
    int a=5;
    int b=6;
    int c=a×b;
    return c;
}
public static double si()
{
    double p=13000;
    double t=2.5;
    double r=12;
    double si=(p×r×t)/100;
    return si;
}
psvm(String[] args)
{
    S.O.P(pro());
    S.O.P(si());
}
```

O/P:
30
3900.0

Chain function call

```
public static int pro()
{
    int a=5;
    int b=6;
    int c=a×b;
    S.O.P(si());
    return c;
}
public static double si()
{
    double p=13000;
    double t=2.5;
    double r=12;
    double si=(p×r×t)/100;
    return si;
```

O/P:-
3900.0
30

```
psvm(String[] args)
{
    S.O.P(pro());
}
```

With parameter and with returntype

WAP to print product of 2 numbers

```
class Product
{
    public static int pro(int a, int b)
    {
        int c=a*b;
        return c;
    }
    psvm(String[] args)
    {
        S.O.P(pro(5,6));
        S.O.P(pro(15,16));
    }
}
```

O/P:-
30
240

2) WAP to print simple interest

```
class Product
{
    public static double si(double p, double t, double r)
    {
        double si=(p*r*t)/100;
        return si;
    }
    psvm(String[] args)
    {
        S.O.P(si(13000,2.5,13));
        S.O.P(si(10000,4,3));
    }
}
```

O/P:
3900.0
1200.0

Question:-

1) Write a java program by following the below scenario's
   a) Create a class called as addition
   b) Declare a static method called as add which accepts three arguments and return the sum of those 3 arguements
   c) Underneath main method call the add method by passing necessary values and print the sum of those 3 arguements

```
class Addition
{
  public static int add (int a, int b, int c)
  {
    int sum = a+b+c; // return a+b+c;
    return sum;
  }
  public static void main (String[] args)
  {
    System.out.println("The sum of above three numbers is:" + add(3,4,5))
  }
}
```

1) without arguement without return type

```
class Demo
{
  public static void display()
  {
    S.O.P("GM");
  }
```

2) with arguements without return type

```
public static void show(int age, String name)
{
  S.O.P(age+" "+name)
}
```

3) without arguement with return type

```
static int m1()
{
  return 10;
}
psvm()
{String x = m1();
 S.O.P(x);
}
```

4) with arguement with return type

```
static double m2(int x)
{
  return x+10.5;
}
psvm()
{
  S.O.P(m2(10));
}
```

Note:-

1) If a method is returning some value we have to either store it or print it.

Q2) Write a java program to print the multiples of 5 from 1 to 25 in reverse order using while loop.

Q3) Write a java program to find the largest of 3 numbers

Q2 Sol:

```
class MultipleOfFive
{
  public static void main(String[] args)
  {
    int n = 25;
    while(n >= 1)
    {
      S.O.P(n);
      n=n-5;
    }
  }
}
```

Q) class LargestOfThreeNumbers

Sol:
```
{
    psvm ()
    {
        int a = 10;
        int b = 20;
        int c = 30;
        if(a > b && a > c)
        {
            S.O.P("a is largest");
        }
        else if(b > a && b > c)
        {
            S.O.P("b is largest");
        }
        else
        {
            S.O.P("c is largest");
        }
    }
}
```

or

```
class Largest
{
    psvm ()
    {
        int a = 10;
        int b = 20;
        int c = 30;
        int largest = (a>b)?(a>c?a:c):
        S.O.P(largest);
        int largest = (a>b)?(a>c?a:c):
                      (b>c?b:c);
        S.O.P("largest among "+a+",
              +b+" and "+c+" is "+
              largest);
    }
}
```

Factorial:-

```
int fact = 1;
int res = 10;
for(int i = 1; i<res; i++)
{
    fact = fact * i;
}
S.O.P(res)
```

Object Oriented Programming

Object:-

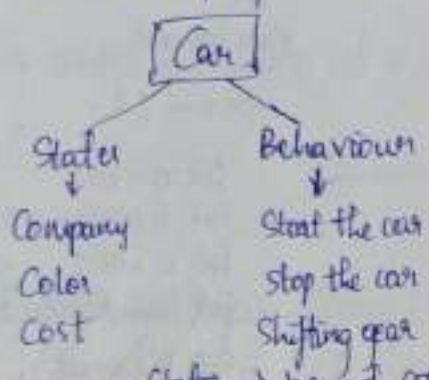
* Anything which is present in real world and physically existing is called as object.
* Object is also referred as real world entity.

Ex:- Car, laptop, Mobile, Fan etc.,

* Every object will have certain properties and those properties are categorized into 2 types.
  - → States
  - → Behaviour
* States are the properties which is used to store some data

* Behaviour are the properties used to perform some task.

Ex:- 1)



2) Mobile
- States → brand, cost, modelno
- Behaviour → call, text, pics

Note:- In java object name is referred as class name, states are referred as variables/data members, behaviour is referred as method/member function.

class:-

* class is a blueprint of an object.
* class is also a platform to store states and behaviours of an object
* class should be declared using keyword 'class'

Syntax:- class ClassName

```
{
    //variables
    //Methods
    //Blocks
    //Constructor
}
```

Scope of class

Ex:- 1) class Car
```
{
}
```

2) class Student
```
{
}
```

WIP:-

Object Creation:-

* Object Creation is a process to store non static members (variables & methods) inside memory.
* Objects will get stored inside a memory location called as "heap area".
* new operator is responsible for creating an object internally.

Syntax for object creation

ClassName reference name /object name = new ClassName();

Ex:- 1) class Car
```
{
}
```
⇒ Car c = new Car();

2) class Pen
{ => Pen p = new Pen();
}

Note:- In order to access a non static variable we make use of the below syntax.

objectname.variablename

Q) How to identify a static variable and a non static variable?

* If a variable is declared using static keyword, we refer it as static variable.

Ex:- static int age = 10;

* If a variable is not declared using static keyword, we refer it as non static variable.

Ex:- int age = 10;

Program:-

```
package jspiders;
class Student
{      Non static variables
    int age = 22;
    String name = "Tom";
    public static void main(String[] args)
    {
        Student s = new Student (); //object creation
        System.out.println("Age is "+s.age);
        System.out.println("Name is "+s.name);
    }
}
```

O/P:- Age is 22
Name is Tom

```
class Employee
{
    int Id = 102;
    String name = "Jerry";
    psvm()
    {
        Employee emp = new Employee();
        S.O.P(emp.id+" "+emp.name);
        S.O.P("Emp Id of "+emp.name+"is"+emp.id);
```

```
        S.O.P("Id:" + emp.id + "\nName: " + emp.name);
    }
}
```

O/P:- 102 Jerry
EmpId of Jerry is 102
Id: 102
Name: Jerry

Write a java program by following the below scenario
1) create a class called mobile
2) declare & initialize two variable called as brand & cost.
3) under main method create a instance or object of mobile and print the brand & cost.

```
class Mobile
{
    int cost = 70000;
    String brand = "Apple";
    psvm()
    {
        Mobile m = new Mobile();
        S.O.P("The" + m.brand + " costs about" + m.cost);
    }
}
```

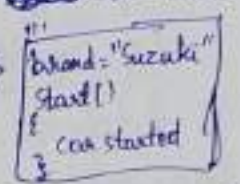
Note:- Non static members (variables & methods) can be accessed either in same class or different class with the help of Object creation

Syntax:-
→ objectname.variablename
→ objectname.methodname()

Ex:-
```
class Car
{
    String brand = "Suzuki";
    void start()
    {
        S.O.P("Car started");
    }
}
```

```
class Test
{
  psvm()
  {
    Car c = new Car();
    S.O.P("Brand is"+c.brand);
    c.start();
  }
}
```

O/P:- Brand is Suzuki
Car Started

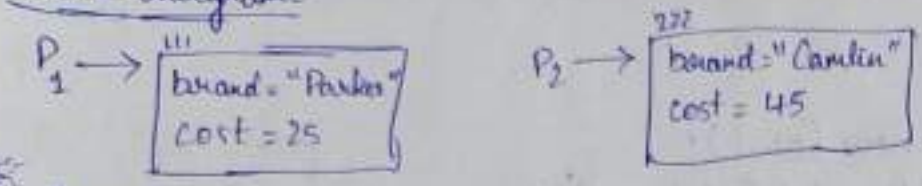


Note:- * It is possible to create multiple objects for a specific class
* When we create multiple objects, it simply means we are creating multiple copies or multiple instances of that class.
* Changes made to one object does not affect another object.

Ex:-
```
class Pen
{
  String brand = "Camlin";
  int cost = 25;
  psvm()
  {
    Pen p1 = new Pen();
    Pen p2 = new Pen();
    S.O.P(p1.brand + " " + p1.cost);
    S.O.P(p2.brand + " " + p2.cost);
    p1.brand = "Parker";
    p2.cost = 45;
    S.O.P(p1.brand + " " + p1.cost);
    S.O.P(p2.brand + " " + p2.cost);
  }
}
```

O/P:- Camlin 25
Camlin 25
Parker 25
Camlin 45

Block diagram:-



Ex ① class Person

```
{
    int age = 21;
    String name = "Ullas";
    void walk()
    {
        S.O.P ("The person is walking");
    }
    void eat()
    {
        S.O.P("The person is eating");
    }
    psvm()
    {
        Person p = new Person();
        S.O.P("Age: "+p.age+"\n Name: "+p.name);
        p.walk();
        p.eat();
    }
}
```

O/P:- Age : 21
Name : Ullas
The Person is walking
The Person is eating

② class Employee

```
{
    int ID = 07;
    String name = "vishwa";
    void work()
    {
        S.O.P ("The employee is working");
    }
    class Solution
    {
```

```
ps vm()
{
    Employee e = new Employee();
    S.O.P("ID: "+e.ID+"\n Name: "+e.name);
    e.work();
}
}
```

O/P:- ID : 07
Name: vishwa
The employee is working

Default values:-

* If a variable is declared & not initialized to any value. Then the compiler will automatically initialize to its default value.
* Default values are applicable for both static and nonstatic variable

Data types	Memory size Byte	Memory size Bits	Default value	When we use this datatypes
byte short int long	1 2 4 8	8 16 32 64	0	Non decimal Numeric value
float double	4 8	32 64	0.0	Decimal Numerical value
char	2	16	Unicode value Non printable value	Single character (' ')
boolean	X	1 bit	false	true / false
String	X	X	null	Set of characters (" ")

Ex:- class Defaultvalues
```
{
    int a;
    double b;
    char c;
    boolean d;
    String e;
```

```
psvm
{
    DefaultValues d=new DefaultValues();
    S.O.P(d.a);
    S.O.P(d.b);
    S.O.P(d.c);
    S.O.P(d.d);
    S.O.P(d.e);
}
}
```

O/P:- 0
0.0
[ ] empty
false
null

Q1) WAP to store 2 student details where in each student will have 2 attributes called as age and name.

```
class Student
{
    int age;
    String name;
    psvm ()
    {
        Student s1 = new Student();
        s1.age = 21;
        s1.name = "Tom";
        S.O.P(s1.name+" is"+s1.age+" years old");
        Student s2 = new Student();
        s2.age = 22;
        s2.name = "Jerry";
        S.O.P(s2.name+" is"+s2.age+" years old");
    }
}
```

O/P:- Tom is 21 years old
Jerry is 22 years old

Q2) WAP to store 2 employee details & print their respective values where in each employee will have 3 attributes called as ID, name & salary.

Sol

```
class Employee
{
    int id;
    String name;
    double salary;
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
        e1.id = 01;
        e1.name = "john";
        e1.salary = 30000.00;
        e2.id = 02;
        e2.name = "smith";
        e2.salary = 35000.00;
        e3.id = 03;
        e3.name = "allen";
        e3.salary = 40000.00;
        S.O.P(e1.name + "id is: " + e1.id + "and salary is: " + e1.sala
        S.O.P(e2.name + "id is: " + e2.id + "and salary is: " + e2.salary);
        S.O.P(e3.name + "id is: " + e3.id + "and salary is: " + e3.salary);
    }
}
```

O/P:-

```
john id is: 01 and salary is: 30000.00
smith id is: 02 and salary is: 35000.00
allen id is: 03 and salary is: 40000.00
```

Static Keyword:-

* Static is a keyword which can be used with a variable, method, blocks and classes.
* Static members will only have 1 copy per class
* Static members will get stored inside a memory location

called as class Area or Static pool.

* All the static members should be accessed with the help of class Name where in object creation is not necessary.

Syntax:-

→ ClassName. variableName

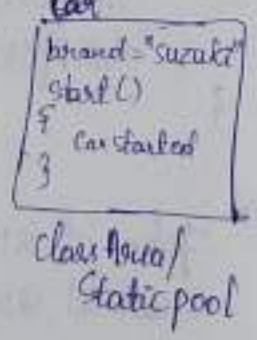
→ ClassName. methodName()

Note:- All the static members (variables & methods) should be declared using static keyword.

Ex:- Accessing static members within the class

```
class Car
{
    static String brand = "Suzuki";
    static void start ()
    {
        S.O.P("Car Started");
    }
    public static void main (String[] args)
    {
        S.O.P (Car.brand);
        Car.start ();
        S.O.P (brand);  //ClassName → Car.brand
        start ();       //ClassName → Car.start()
    }
}
```

O/P:- Suzuki
Car Started
Suzuki
Car started



Accessing static members outside the class

```
class Employee
{
    static int Id = 101;
    static void work()
    {
        S.O.P("Employee is working");
    }
}
```

```
class Test
{
    psvm()
    {
        S.O.P(Employee.id);
        Employee.work();
    }
}
```

O/P:- 101
Employee is working

Q1)

```
class Mobile
{
    static String brand="Apple";
    static int cost = 85000;
    static void playGames()
    {
        S.O.P("Game Started");
    }
    static void bookCab()
    {
        S.O.P("Cab booked");
    }
}
class Solution
{
    psvm()
    {
        S.O.P(Mobile.brand);
        S.O.P(Mobile.cost);
        Mobile.playGames();
        Mobile.bookCab();
    }
}
```

O/P:- Apple
85000
Game Started
Cab booked.

	Accessing in same class	Accessing in different class	Memory location	Number of copies
Static	Direct or ClassName	ClassName	Class Area or Static pool	Only 1 copy
Non Static	Object Creation	Object Creation	Heap Area	Multiple copies depends on no of object creation

Method Overloading:-

* In a class having multiple methods with the same name but difference in arguements is called as method overloading.
* In other words the process of having ~~[illegible]~~ multiple methods with the same name but change in parameters, is called as method overloading.
* In order to achieve method overloading, we have to follow atleast 1 of the following 3 rules.

→ No of arguements (There should be change in the no of arguements)
→ There should be change in the datatype of arguements.
→ There should be change in the sequence ~~of~~ or order of the datatypes

Method overloading for non static methods

Ex:-
```
class Student
{
    void display(int id)
    {
        S.O.P("Id:"+id);
    }
    void display(double height)
    {
        S.O.P("Height:"+height);
    }
    void display(int age, String name)
    {
        S.O.P("Age:"+age+"\nName:"+name);
    }
    void display(String name, int age)
    {
        S.O.P("Name:"+name+"\nAge:"+age);
    }
}
class MainClass
{
    psvm()
    {
        Student s=new Student();
        s.display("Tom", 22);
        s.display(101);
        s.display(21, "Jerry");
        s.display(5.8);
    }
}
```

O/P:- Name : Tom
Age : 22
Id : 101
Age : 21
Name : Jerry
Height : 5.8

Note:- Both static methods & non static methods can be overloaded.

Method overloading for static methods

```
class Flipkard
{
    static void login(String email, int password)
    {
        S.O.P("Logged in using email and password");
        S.O.P("Email : "+email+" & Password "+password);
    }
    static void login(int contactNo, int otp)
    {
        S.O.P("Logged in using contact no and otp");
        S.O.P("Contact No : "+contactNo+" & OTP : "+otp);
    }
    psvm()
    {
        login(98765, 4545);
        S.O.P("--------");
        login("dinga@gulku.com", 123);
    }
}
```

O/P:-

Logged in using contact no and otp
Contact No : 98765 & OTP : 4545

Logged in using email and password
Email : dinga@gulku.com & Password : 123

Note:- In java we can overload even main method but the execution will always begin from the main method which accepts string array as an argument

```
class Demo
{
    public static void main(int x)
    {
        S.O.P("x:"+x);
    }
    public static void main(String[] args)
    {
        S.O.P("start");
        main(1.5);
        main(100);
        S.O.P("end");
    }
    public static void main(double x)
    {
        S.O.P("x:"+x);
    }
}
```

O/P:-
start
x: 1.5
x: 100
end

Scanner:-

* Scanner is a pre defined class present in java.util package
* Scanner class is used to accept the input from the user

Rules for using scanner class:-

1) Create an object of scanner class
2) Pass system.in to the constructor of scanner class

Syntax:-

```
Scanner sc = new Scanner(System.in);
```

3) Import scanner class from java.util package

Syntax:- import java.util.Scanner;

4) Optionally, we can close the scanner object

Syntax:- objectName.close(); → sc.close();

5) Make use of predefined methods to accept specific inputs from the user

Important methods present in scanner class

1) byte → nextByte()
2) short → nextShort()
3) int → nextInt()
4) long → nextLong()

5) float → nextFloat()
6) double → nextDouble()
7) boolean → nextBoolean()
8) String → next() and nextLine() (Assignment)
9) char → next().charAt(0)
"java".charAt(0)

Ex: ① ...

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        S.O.P("Enter First Number:");
        int a = sc.nextInt();
        S.O.P("Enter Second Number:");
        int b= sc.nextInt();
        int sum = a+b;
        S.O.P("SUM :"+sum);
        sc.close();
    }
}
```

o/p:-
Enter First Number
10
Enter Second Number
35
SUM:45

②

```
import java.util.Scanner;
class Employee
{
    public static void main(String[] args)
    {
        Scanner scan=new Scanner(System.in);
        S.O.P("Enter Id:");
        int Id = scan.nextInt();
        S.O.P("Enter Name:");
        String name = scan.next();
        S.O.P("Enter Salary:");
        double salary = scan.nextDouble();
```

```
        S.O.P("Enter Gender:");
        char gender = scan.next().charAt(0);
        scan.close();
        S.O.P("ID:"+id+"\nName:"+name+"\nSalary:"+salary+"
            \nGender:"+gender);
    }
}
```

O/P:- Enter Id:

100
Enter Name:
Jack
Enter Salary:
34000.21
Enter Gender:
M
Id: 100
Name: Jack
Salary: 34000.21
Gender: M

③ Scanner class using method & loop

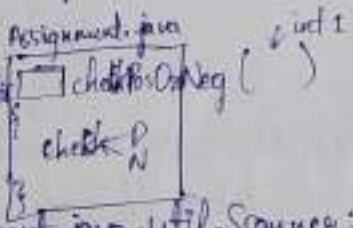
```
import java.util.Scanner;
class Addition
{
    static void add(int x, int y)
    {
        S.O.P("Sum of "+x+" & "+y+" is "+(x+y));
    }
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        for(int i=1; i<=3; i++)
        {
            S.O.P("Enter 2 Numbers:");
            int x = sc.nextInt(); //25
            int y = sc.nextInt(); //35
            add(x,y); //add(25,35)
    }
```

```
        scan.close();
    }
}
```

O/P:- Enter 2 Numbers:

10
20
Sum of 10 & 20 is 30
Enter 2 Numbers:
25
35
Sum of 25 & 35 is 60
Enter 2 Numbers:
100
350
Sum of 100 & 350 is 450

Q1) Step 1: Assignment.java

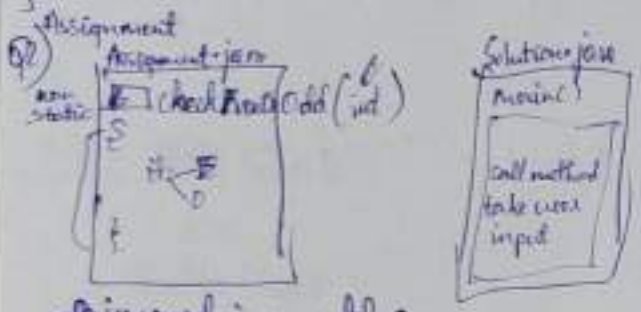


Step 2: Solution.java

```
import java.util.Scanner;
class Assignment
{
    static void checkPosOrNeg (int x)
    {
        if(x>0)
        {
            S.O.P("Number is Positive");
        }
        else
        {
            S.O.P("Number is Negative");
        }
    }
}
class Solution
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        for(int i=1;i<=3;i++)
        {
            S.O.P("Enter Number");
```

```
        int x = sc.nextInt();
        Assignment.checkPosOrNeg(x);
      }
      sc.close();
    }
}
```

Q2)




```
import java.util.Scanner;
class Assignment
{
    void checkEvenOrOdd(int num)
    {
        if(num % 2 == 0)
        {
            S.O.P(num+" is Even number");
        }
        else
        {
            S.O.P(num+" is Odd number");
        }
    }
}
class Solution
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        Assignment a = new Assignment();
        for(int i=1; i<=5; i++)
        {
            S.O.P("Enter number:");
            int num=scan.nextInt();
            a.checkEvenOrOdd(num);
        }
        scan.close();
    }
}
```

Blocks:-

* Blocks are a set of instructions or a block of code which is used for initialization.
* Blocks are categorized into 2 types
  → Static Blocks
  → Non Static Blocks

Static Blocks:-

* Static Blocks are a set of instructions which is used for initializing static variables.
* Static Blocks will get executed during class loading Time (before main() method)

*Syntax:-

```
static
{
  ≡≡≡
}
```

* If there are multiple static blocks, the execution will always happen in sequential order.

Ex:-

```
class Tom
{
    static
    {
        S.O.P("In static block-1");
    }
    psvm()
    {
        S.O.P("Hello");
    }
    static
    {
        S.O.P("In static block-2");
    }
    static
    {
        S.O.P("In static block-3");
    }
}
```

O/P:-

```
In static block-1
In static block-2
In static block-3
Hello
```

```
class Student
{
    static int age;  //0 -> 20 -> 30
    static
    {
        S.O.P("Initializing age to 20");
        age=20;
    }
    psvm()
    {
        S.O.P("AGE:"+Student.age);
    }
    static
    {
        S.O.P("Initializing age to 30");
        age=30;
    }
}
```

o/p:-
Initializing age to 20
Initializing age to 30
AGE:30

0 -> 20 -> 30

Non Static Blocks (Instance Block):-

* Non Static Blocks are a set of instructions which is used for initializing both static & non static variables.
* Non static blocks will always get executed at the time of object creation.
* Syntax:-

```
{
}
```

* Non static also called as Instance Block.
* If there are multiple non static blocks, the execution will always happen in sequential order.

Ex:-

```
class Car
{
    {
        S.O.P("In Non-Static Block-1");
    }
    psvm()
    {
        S.O.P("Start");
        Car C = new Car();
        new Car();
        S.O.P("End");
    }
    {
        S.O.P("In Non-Static Block-2");
    }
```

O/P:- Start
In Non-Static Block-1
In Non-Static Block-2
In Non-Static Block-1
In Non-Static Block-2
End

Ex:- class Employee

```
{
  int id;   //0 -> 101 -> 201
  { //instance block
    //S.O.P("ID:"+id);
    id=101;
    //S.O.P("ID:"+id);
  }
  psvm()
  {
    Employee emp = new Employee();
    S.O.P("ID:"+emp.id);
  }
  { //Instance block
    id=201;
  }
}
```

O/P:-
ID:201

Ex:- class Pen

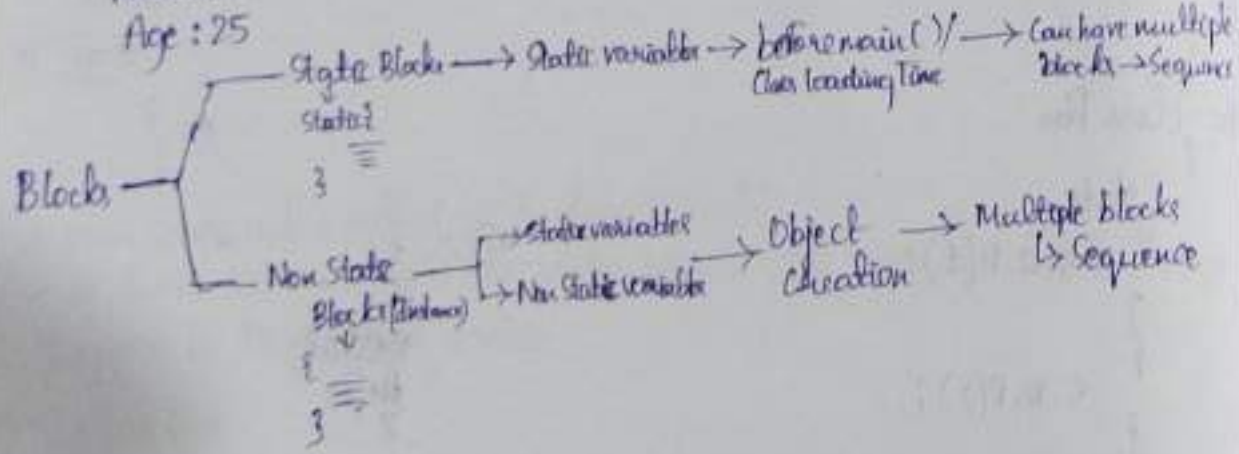
```
{
  static
  {
    S.O.P(1);
  }
  {
    S.O.P(2);
  }
  static
  {
    S.O.P(3);
  }
  psvm()
  {
    Pen p=new Pen();
    S.O.P("Hello");
    ~~Pen p=new Pen();~~
    new Pen();
  }
}
```

O/P:- 1
3
2
Hello
2

```
class Person
{
    static int age;
    String name;
    static
    {
      age = 22;
    }
    {
      age = 25;
      name = "Tom";
    }
    psvm()
    {
      S.O.P("Age:" + Person.age); //22
      Person p = new Person();
      S.O.P("Name:" + p.name); //Tom
      S.O.P("Age:" + Person.age); //25
    }
}
```

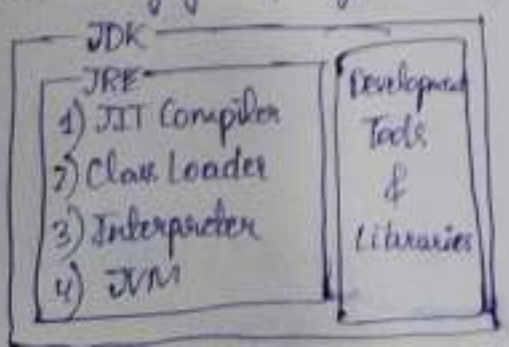
O/P:-

Age : 22
Name : Tom
Age : 25



JDK:-

* JDK stands for Java Development kit
* JDK is a software which contains all the resources for developing and executing java programs.



JRE :-

* JRE stands for Java Runtime Environment.
* JRE is a software which provides a platform for executing java programs.

JIT Compiler:-

* JIT stands for Just In Time Compiler.
* JIT compiler converts byte code into machine understable language.

Class loader:-

* Class loader is used to load the class from secondary storage to executable area.

Interpreter:-

Interpreter is used to execute the program line by line frequentially

JVM:-

* JVM stands for Java Virtual Machine
* JVM is the manager of the JRE

JVM Architecture:-

1) Heap Area:-

It is a memory location to store all the non static members.

2) Class Area:-

It is a memory location to store all the static members.

3) Method Area:-

The method implementation is stored inside method area.

4) Stack:-

The execution of a program happens inside a stack.

Constructors:-

* Constructor is a set of instructions or a block of code which is used for initialization and instantiation.
* Initialization means assigning values or storing values.
* Instantiation means Object Creation.
* Class Name & Constructor Name should be same.
* Constructors are also referred as special methods which does not have return type.
* Constructors will get executed at the time of Object Creation

Ex Syntax:- Class Name ( ) {

}

Ex: ① Non Parameterized

```
Class Car
{
    Non parameterized custom/user defined constructor
    Car()
    {
        S.O.P("In Car");
    }
    psvm()
    {
        S.O.P("Start");
        new Car();
        S.O.P("End");
    }
}
```

O/P:- Start
In Car
End

② Parameterized

```
class Person
{
    Parameterized custom/user-defined constructor
    Person(int age, String name)
    {
        S.O.P("Age:" +age+ "\n Name:" +name);
    }
    psvm()
    {
        Person p= new Person(25, "Jack");
    }
}
```

O/P:-
Age: 25
Name: Jack

Types of Constructors:-

* Default Constructor
* Custom or User-Defined Constructor

Default Constructor:-

* If a class does not have any explicit constructor, then the compiler will automatically generate a constructor, this constructor is referred as default constructor.
* Default constructors does not accept any arguements and does not have any implementation.
* Default constructors are implicitly present.

Custom/User-Defined Constructor:-

* If a constructor is explicitly present inside a class, then we refer it as custom constructor.
* Custom constructors are generated by the user or the programmer.
* Custom constructor can be categorized into 2 types
  → Non Parameterized Custom constructor
  → Parameterized Custom Constructor.

Note:- Both default constructor & custom constructor cannot be present together

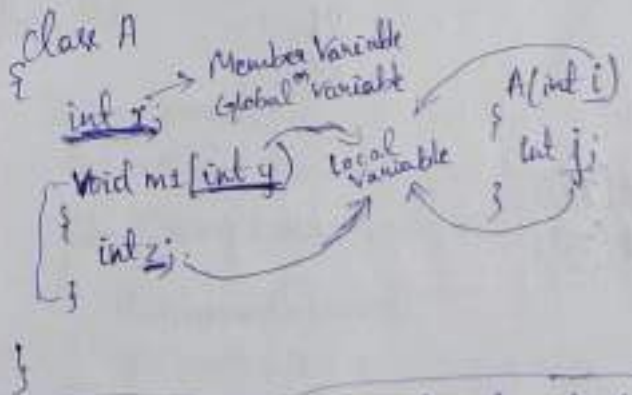
Member Variables & Local Variables

Member Variables:-

* If a variable is declared within a class we call it as member variable
* Member variables can be accessed globally (everywhere).
* Member variables are categorized in 2
  → Static &
  → Non static variables

Local Variables:-

* If a variable is declared within a specific scope or block, we refer it as local variables.
* Local variables are accessible only within that specific block.



Note: * Default values are applicable only for member variables (static & non-static variables).
* Default values are not applicable for local variables, therefore local variables should be initialized before using it.

this keyword:-

* In java if member variables and local variables names are same, local variables will dominate the member variables, therefore to avoid this confusion we come across 'this' keyword
* 'this' keyword is used to point to the current object or current instance.

```
class Kangaroo
{
    double height = 5.5; //Member Variable
    void display()
    {
        double height = 4.4; //local Variable
        S.O.P(height); //4.4
        S.O.P(this.height); //5.5
    }
    psvm()
    {
        Kangaroo k = new Kangaroo();
        k.display();
    }
}
```

O/P:-
4.4
5.5

Ex ①

```
class Student
{
    int age; //0 21
    Student(int age)
    {
        this.age = age;
    }
    psvm()
    {
        Student s1 = new Student(21);
        Student s2 = new Student(22);
        S.O.P(s1.age);
        S.O.P(s2.age);
    }
}
```

O/P:-
21
22

s1 → 111 | s2 → 222

age = ~~0~~ 21	age ~~0~~ 22

```
class Employee
{
    int id;
    String name;
    Employee(int id, String name)
    {
        this.id = id;
        this.name = name;
    }
    psvm()
    {
        Employee e1 = new Employee(101, "Tom");
        S.O.P("Id:"+e1.id+"\nName:"+e1.name);
        S.O.P("------");
        Employee e2 = new Employee(201, "Jack");
        S.O.P("Id:"+e2.id+"\nName:"+e2.name);
    }
}
```

O/P:-

```
Id:101
Name:Tom
Id:201
Name:Jack
```

Q1) Write a java program by following the below sentences.

* Create a class called as Bike
* Declare 2 attributes called as brand & cost
* Initialize those attributes with the help of a contructor
* Under main method 2 instances of Bike & print their respective brand & cost

```
class Bike
{
    String brand;
    int cost;
    Bike(String brand, int cost)
    {
        this.brand = brand;
        this.cost = cost;
    }
    psvm()
    {
        Bike b1 = new Bike("Duke", 250000);
        S.O.P("Brand:"+b1.brand+"\n Cost:"+b1.cost);
        Bike b2 = new Bike("Ducati", 450000);
        S.O.P("Brand:"+b2.brand+"\n Cost:"+b2.cost);
    }
```

O/P:-

```
Brand:Duke
Cost:250000
Brand:Ducati
Cost:450000
```

5.

```
import java.util.Scanner;
class Student
{
    int id;
    String name;
    double marks;
    Student (int id, String name, double marks)
    {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }
    psvm(....)
    {
        Scanner sc = new Scanner(System.in);
        S.O.P("Enter id, name, marks:");
        int id = sc.nextInt();
        String name = sc.next();
        double marks = sc.nextDouble();
        Student s = new Student (id, name, marks);
        S.O.P("Id:"+s.id+"\n Name:"+s.name+"\n Marks:"+s.marks);
        sc.close();
    }
}
```

O/P:-

```
Enter id, name, marks
101
Jack
89.28
Id: 101
Name: Jack
Marks: 89.28
```

6) Write a java program by following the below scenario's.

* Create a class called as laptop
* Declare 3 attributes called as brand, cost, ramSize.
* Initialize all 3 attributes with the help of a constructor
* Create another class called as Solution and under main method create 2 objects of laptops and print their respective details

```
~~import java.util.Scanner;~~
class laptop
{
    String brand;
    int cost;
    int ramsize;
    laptop (String brand, int cost, int ramsize)
    {
        this.brand = brand;
```

```
        this.cost = cost;
        this.ramsize = ramsize;
    }
}
class Solution
{
    psvm(----)
    {
        Laptop l1 = new Laptop("ASUS", 77000, 16);
        Laptop l2 = new Laptop("HP", 70000, 8);
        S.O.P("Brand:"+l1.brand+"\n Cost:"+l1.cost"\n RamSize:"+l1.ramsize
        S.O.P("Brand:"+l2.brand+"\n Cost:"+l2.cost"\n RamSize:"+l2.ramsize);
    }
}
```

Ex:-

```
class Country
{
    String name;
    int population;
    Country(String name, int population)
    {
        this.name = name;
        this.population = population;
    }
    void display
    {
        S.O.P("Country Name:"+name);
        S.O.P("Population in Crores:"+population);
    }
}
class Solution
{
    psvm(----)
    {
        Country c1 = new Country("India", 135);
        Country c2 = new Country("USA", 35);
        Country c3 = new Country("Sri Lanka", 2);
        S.O.P("Country Details:");
        S.O.P("--------");
        c1.display();
        S.O.P("--------");
```

```
c2.display();
S.O.P("----------");
c3.display();
S.O.P("==========");
S.O.P("Country Name:"+c1.name);
S.O.P("Population:"+c1.population);
S.O.P("----------");
S.O.P("Country Name:"+c2.name);
S.O.P("Population:"+c2.population);
S.O.P("----------");
S.O.P("Country Name:"+c3.name);
S.O.P("Population:"+c3.population);
}
}
```

O/P:

```
Country Details
----------
Country Name: India
Population in Crores: 135
----------
Country Name: USA
Population in Crores: 35
----------
Country Name: Sri Lanka
Population in Crores: 2
==========
Country Name: India
Population: 135
----------
Country Name: USA
Population: 35
----------
Country Name: Sri Lanka
Population: 2
```

Constructor Overloading:-

* The process of having multiple constructors in a class but difference in arguements is called as Constructor Overloading.

* To achieve constructor overloading we have to follow the below rules

→ There should be change in number of arguements

→ There should be change in datatype of arguements

→ There should be change in the sequence of ~~argue~~ datatypes

Ex:

```
class Employee
{
    Employee(int id)
    {
        S.O.P("Id:"+Id);
    }
    Employee(String name, int age)
    {
        S.O.P(name+"is"+age+"years old");
    }
    Employee(double salary)
    {
        S.O.P("Salary:"+salary);
    }
    Employee(int age, String name)
    {
        S.O.P("Age:"+age+"ft Name:"+name);
    }
}
class MainClass
{
    psvm()
    {
        Employee e1 = new Employee(10123.05);
        new Employee(101);
        new Employee("Tom", 23);
        Employee e2 = new Employee(25, "Jack");
    }
}
```

O/P:-

Salary: 10123.05

Id: 101

Tom is 23 years old

Age: 25 Name: Jac

Array:-

* Array is a container in order to store a group of data or elements
* Array is homogeneous in nature.
* Array is fixed size
* Array is indexed base and index position starts from 0.

① Array Declaration:-

Syntax:- datatype[] arrayName;

Ex:→ int[] a;

→ double[] x;

② Array Creation:-

Syntax:- arrayName = new datatype[size];

Ex:→ a = new int[4];

→ x = new double[3]

a [0 | 0 | 0 | 0]
0 1 2 3

x [0.0 | 0.0 | 0.0]
0 1 2

③ Array Declaration and Creation:-

Syntax:- datatype[] arrayName = new datatype[size];

Ex:- → int[] a = new int[3];

→ String[] s = new String[2];

a [0 | 0 | 0]
0 1 2

s [null | null]
0 1

④ Array Initialization:-

Syntax:- arrayName[index] = value;

Ex:- → a[1] = 50;

→ s[0] = "Tom";

a [0 | 50 | 0]
0 1 2

s [null | null]
0 1

⑤ Array Declaration & Initialization:-

Syntax:- datatype[] arrayName = {$v_1, v_2, \ldots, v_n$};

Ex:- → int[] a = {10, 20, 30};

a[0] a[1] a[2]

a [10 | 20 | 30]
0 1 2

To know the length of array

Syntax:-

arrayName.length

a.length ⇒ 3

Ex:-

```
class Demo
{
    psvm(----)
    {
        //Array Declaration
        int[] a;

        //Array Creation
        a = new Int[3];

        //Printing Array Elements
        S.O.P(a[0]);
        S.O.P(a[1]);
        S.O.P(a[2]);
        S.O.P("--------");
        //Array Initialization
        a[0] = 25;
        a[2] = 45;
        S.O.P(a[0]);
        S.O.P(a[1]);
        S.O.P(a[2]);
        S.O.P("LENGTH:"+a.length);
        S.O.P("*********");
        //Array Declaration and Creation
        double[] marks = new double[2];
        S.O.P(marks[0]);
        S.O.P(marks[1]);
        S.O.P("--------");
        marks[0] = 25.6;   //Again Array Initialization
        marks[1] = 45.7;
        S.O.P(marks[0]);
        S.O.P(marks[1]);
        S.O.P("LENGTH:"+marks.length);
    }
}
```

O/P:-

```
0
0
0
--------
25
0
45
LENGTH=3
*********
0.0
0.0
--------
25.6
45.7
LENGTH=2
```

5.

```
class Runner
{
    psvm(------)
    {
        int[] a = {10,20,30};
        for(int i=0; i<a.length; i++)
        {
            S.O.P(a[i]);
        }
        S.O.P("--------");
        for(int i=a.length-1; i>=0; i--)
        {
            S.O.P(a[i]);
        }
        S.O.P("*******");
        String[] subjects={"java","apti","web","sql"};
        for(int i=0; i<subjects.length; i++)
        {
            S.O.P(subjects[i]);
        }
        S.O.P("--------");
        for(int i=subjects.length-1; i>=0; i--)
        {
            S.O.P(subjects[i]);
        }
    }
}
```

O/P:

```
10
20
30
--------
30
20
10
*******
java
apti
web
sql
--------
sql
web
apti
java
```

Q) WA java program to find the sum and average of n Array elements

```
class SumAvg
{
    psvm(--------)
    {
        int sum=0;
        int[] a={10,20,30,40};
        for(int i=0;i<a.length;i++)
        {
            sum=sum+a[i];
        }
        S.O.P("SUM:"+sum);
        S.O.P("Average:"+(sum/a.length));
    }
}
```

O/P:-
SUM: 100
Average: 25

Assignment Questions:

1) WA java P to print even numbers in an array.
2) WAP to print odd numbers in an array.
3) WAP to print the multiples of 5 from an array.
4) WAP to find the smallest element in an array.
5) WAP to find the largest element in an array.

Note:- Valid array declarations in java

- int[] a;
- int a[];
- int []a;

Assignment Questions Solution

```
class Even
{
    psvm(.....)
    {
        int[] a={1,2,5,7,8,10,13,17,18,20,23,24};
        for(int i=0;i<a.length;i++)
        {
            if(a[i]%2==0)
            {
                ~~S.O.P(a[i]+" is even number");~~
                S.O.P(a[i]);
            }
        }
    }
}
```

O/P:-
2
8
10
18
20
24

2) class Odd

```
{
    psvm(---)
    {
        int[] a={1,2,5,7,8,10,13,17,18,20,23,24};
        for(int i=0; i<a.length; i++)
        {
            if(a[i]%2==0)
            {
                S.O.P(a[i]);
            }
        }
    }
}
```

O/P:-
1
5
7
13
17
23

3) class MultiplesOfFive

```
{
    psvm(-----)
    {
        int a[]={1,2,5,7,8,10,13,15,17,18,20,23,24,25};
        for(int i=0; i<a.length; i++)
        {
            if(a[i]%5==0)
            {
                S.O.P(a[i]);
            }
        }
    }
}
```

O/P:-
5
10
15
20
25

4) class Smallest

```
{
    psvm(----)
    {
        int a[]={25,13,44,110,70,10,75,5};
        int min = a[0];
        for(int i=1; i<a.length; i++)
        {
            if(a[i]<min)
            {
                min=a[i];
            }
        }
        S.O.P("Smallest number is : "+min);
    }
}
```

O/P:-
Smallest number is : 5

5) class Largest (Program to find largest no in array)

```
{
  psvm(-----)
  {
    int[] a={95,73,44,110,74,80,75,5};
    int max=a[0];
    for(int i=1; i<a.length; i++)
    {
      if(a[i]>max)
      {
        max=a[i];
      }
    } s.o.p("largest number is:"+max);
  }
}
```

O/P:
largest number is: 110

Write a java program by following below scenarios

* Create a class called as student
* Declare 2 attributes called as age & name
* Initialize those 2 attributes with the help of a constructor
* Create another class called as Solution & under main method, create two instances of student, store it inside an array and traverse it using for loop.

```
class Student
{
  Int age;
  String name;
  Student(int age, String name)
  {
    this.age=age;
    this.name=name;
  }
}
class Solution
{
  psvm(-----)
  {
    Student s1=new Student(22,"Emi");
    Student s2=new Student(25,"Jack");
    Student[] s={s1,s2};
    for(int i=0; i<s.length; i++)
    {
```

```
S.O.P (s[i].age + " " + s[i].name);
    }
  }
}
```

Note:- * A class can act as a datatype, if a class acts as a datatype we refer it as non primitive datatype

* Default value for non primitive datatype or class type is 'null'.

Ex:- Car[]c = new Car[2];

c [null | null]

c[0] = 1;
c[1] = 2;

Q) Write a java program by following the below scenarios

* Create a class called as Employee.
* Declare 3 attributes called as id, name, salary.
* Initialize those 3 attributes with the help of a constructor.
* Create another class called as Main class, under main method create 3 instances of employee, store it inside array & traverse it using for loop by printing id, name, salary.

```
class Employee
{
    int id;
    String name;
    double salary;
    Employee (int id, String name, double salary)
    {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
}
class MainClass
{
    p s v m (- - - -)
    {
        Employee e1 = new Employee(101, "Smith", 29000.00);
        Employee e2 = new Employee(201, "Allen", 34000.00);
        Employee e3 = new Employee(301, "Ward", 35000.00);
```

```
        Employee[] e={e1,e2,e3};
        for(int i=0; i<e.length; i++)
        {
            S.O.P(e[i].id+"\t"+e[i].name+"\t"+e[i].salary);
        }
    }
}
```

o/p:-

101	Smith	29000.00
201	Allen	34000.00
301	Ward	35000.00

Note:- In java we can have multiple classes in a single source file but main method should be present within the class which is the file name.

Shortcut: Alt+shift+5

Ex:-

```
class Hotel
{
    String name;
    double rating;
    Hotel(String name, double rating)
    {
        this.name=name;
        this.rating=rating;
    }
    void display()
    {
        S.O.P("Name:"+name);
        S.O.P("Ratings:"+rating);
    }
}
class Runner
{
    P S V M(------)
    {
        Hotel h1=new Hotel("Taj",4.8);
        Hotel h2=new Hotel("Shanthi Sagar",3.9);
        Hotel h3=new Hotel("Sri Ranga",4.0);
        Hotel[] h={h1,h2,h3};
        for(int i=0; i<h.length; i++)
        {
            S.O.P("Name:"+h[i].name+"\nRatings:"+h[i].rating);
            S.O.P("--------");
        }
```

```
        S.O.P("------------");
        for(int i=0; i<h..length; i++)
        {
            h[i].display();
            S.O.P("----------");
        }
    }
}
```

O/P:-

```
Name : Taj
Ratings : 4.8
-----------
Name : Shanthi Sagar
Ratings : 3.9
-----------
Name : Sri Ranga
Ratings : 4.0
**********
Name : Taj
Rating : 4.8
-----------
Name : Shanthi Sagar
Ratings : 3.9
-----------
Name : Sri Ranga
Ratings : 4.0
```

Q) Write a java program to store 4 country objects into an array and print the name of the country with the ~~[illegible]~~ highest population.

~~Program to find~~

```
class Country
{
    String name;
    int population;
    Country(String name, int population)
    {
        this.name = name;
        this.population = population;
    }
    void display()
    {
        S.O.P("Country Name : "+name);
        S.O.P("Population ~~in Crores~~ : "+population);
        S.O.P("**********");
    }
}
```

```
class Solution
{
    P S V M (:----)
    {
        Country C1 = new Country("India", 130);
        Country C2 = new Country("USA", 35);
        Country C3 = new Country("China", 160);
        Country C4 = new Country("Sri Lanka", 2);
        Country C5 = new Country("Russia", 50);
        Country[] C = {C1, C2, C3, C4, C5};
        Country max = C[0];
        for (int i = 0; i < C.length; i++)
        {
            if (C[i] > max)
            {
                max = C[i];
            }
        }
        S.O.P("Country having highest population" + max.display);
        {
            S.O.P(C[i].display);
            S.O.P("----------");
        }
        Country max = C[0];
        for (int i = 1; i < C.length; i++)
        {
            if(C[i].population > max.population)
            {
                max = C[i];
            }
        }
        S.O.P("Highest Population:");
        S.O.P("Name:" + max.name);
        S.O.P("Population:" + max.population);
    }
}
```

O/P:-

Country Name: India
Population: 130

Country Name: USA
Population: 35

Country Name: China
Population: 160

Country Name : Sri Lanka
Population : 2

Highest Population:
Name : China
Population : 160

Assignment Questions :-

q) Write a java program to find the name of the student with the highest marks from an student array.

q) Write a java program to find the name of the employee with the lowest salary from an employee array.

Sol 1)
```
class Student
{
    int id;
    String name;
    int marks;
    Student (int id, String name, int marks)
    {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }
    void marks ()
    {
        S.O.P ("Id:"+id+"\n Name:"+name+"\n Marks:"+marks);
    }
}
class Result
{
    p s v m (----)
    {
        Student S1 = new Student (100, "Aarav", 80);
        Student S2 = new Student (111, "Amit", 87);
        Student S3 = new Student (121, "Varun", 95);
```

```
Student s4=new Student(113,"Vainidhi",65);
Student[] s={s1,s2,s3,s4};
for(int i=0;i<s.length;i++)
{
    s[i].marks();
    S.O.P("----------");
}
Student max=s[0];
for(int i=1;i<s.length;i++)
{
    if(s[i].marks>max.marks)
    {
        max=s[i];
    }
}
S.O.P("**********");
S.O.P("Highest Marks:");
S.O.P("Id:"+max.id+"\nName:"+max.name+"\nMarks:"+max.ma
}
}
```

O/P:-

```
Id:100
Name:Aarav
Marks:80
----------
Id:111
Name:Aniket
Marks:67
----------
Id:121
Name:Varun
Marks:95
----------
Id:131
Name:Vainidhi
Marks:65
**********
Highest Marks
Id:121
Name:Varun
Marks:95
```

```
class Employee
{
    int id;
    String name;
    double salary;
    Employee(int id, String name, double salary){
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    void display(){
        S.O.P("Id:"+id+"\n Name:"+name+"\n Salary:"+salary);
    }
}
class MinSalary
{
    p s v m(-----)
    {
        Employee e1 = new Employee(101, "Tom", 33000);
        Employee e2 = new Employee(111, "Smith", 36000);
        Employee e3 = new Employee(121, "Jerry", 30000);
        Employee e4 = new Employee(131, "Jack", 39000);
        Employee[] e = {e1, e2, e3, e4};
        for (int i = 0; i < e.length; i++){
            e[i].display();
            S.O.P("----------");
        }
        Employee min = e[0];
        for (int i = 1; i < e.length; i++){
            if(e[i].salary < min.salary){
                min = e[i];
            }
        }s.o.p("**********");
        S.O.P("Lowest Salary:");
        S.O.P("Id:"+min.id+"\n Name:"+min.name+"\n Salary:"+min.salary
    }
}
```

O/P:

Id: 101
Name: Tom
Salary: 33000

Id: 111
Name: Smith
Salary: 36000

Id: 121
Name: Jerry
Salary: 30000

Id: 131
Name: Jack
Salary: 39000

Lowest Salary:
Id: 121
Name: Jerry
Salary: 30000

Inheritance:-

* The process of one class acquiring the properties of another class is called as inheritance.
* A class which is sharing the properties is called as Super class or Parent class or Base class.
* A class which is acquiring the properties is called as Sub class or Child class or Derived class.
* Inheritance is achieved with the help of extends keyword.
* Inheritance is also referred as "IS-A-Relationship".
* In java only variables & methods are inherited where as blocks and constructors are not inherited.

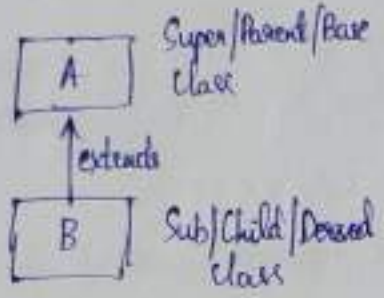
Types of inheritance:-

* Single level Inheritance
* Multi level Inheritance
* Hierarchical Inheritance
* Multiple Inheritance
* Hybrid Inheritance

Note:- * Java explicitly supports only single level, multi level & Hierarchical
* Java doesnot support multiple Inheritance.

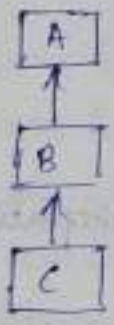
① Single level Inheritance:-

A combination of one super class and one sub class is called as Single level Inheritance.



② Multi level Inheritance:-

A combination of multiple super class and multiple sub class in a sequential order is called as Multi level Inheritance.

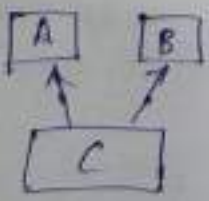


③ Hierarchical Inheritance:-

A combination of one super class having multiple sub classes is called as Hierarchical Inheritance.

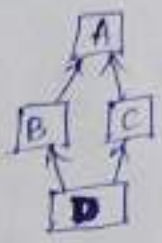
④ Multiple Inheritance:-

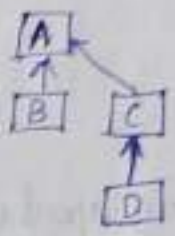
A combination of multiple super class having one sub class is called as Multiple Inheritance



⑤ Hybrid Inheritance:-

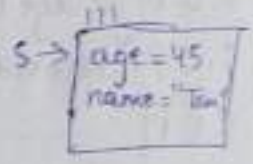
A combination of two or more type of inheritance is called as Hybrid Inheritance.





Ex:- 1) Single level Inheritance

```
class Father{
    int age = 45;
}
class Son extends Father{
    String name = "Tom";
    psvm(----)
    {
        Son s = new Son();
        S.O.P(s.age);
        S.O.P(s.name);
    }
}
```



O/P:-

45
Tom

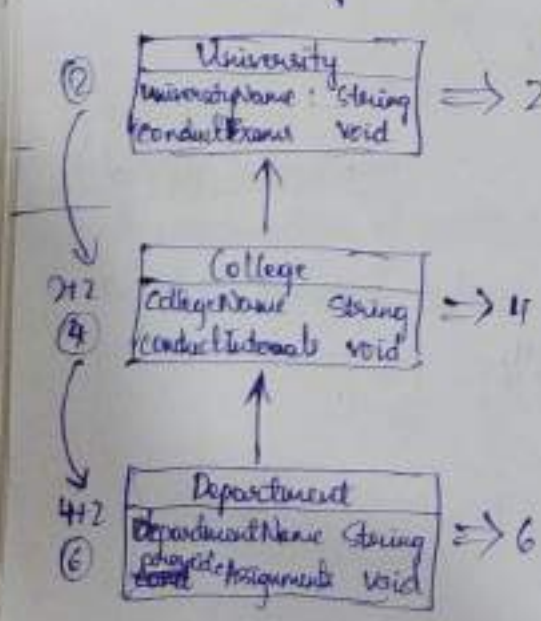
2) Multi Level Inheritance

```
class University{
    String universityName = "VTU";
    void conductExams(){
        S.O.P("VTU Conducting exams");
    }
}
class College extends University{
    String collegeName = "BIT";
    void conductInternals(){
        S.O.P("BIT Conducting internals");
    }
}
class Department extends College{
    String departmentName = "CSE";
    void provideAssignments(){
```

```
        S.O.P("CSE dept given some assignments");
    }
}
class MainClass
{
    p s v m (- - - -){
        Department d = new Department();
        S.O.P("University Name:"+d.universityName);
        S.O.P("College Name:"+d.collegeName);
        S.O.P("Department Name:"+d.departmentName);
        S.O.P("----------");
        d.provideAssignments();
        d.conductInternals();
        d.conductExams();
    }
}
```

O/P:-

```
University Name : VTU
College Name : BIT
Department Name : CSE
-----------------
CSE dept given some assignments
BIT Conducting Internals
VTU Conducting Exams
```



Assignment:- 1) Hierarchical Inheritance

```
class Vehicle{
    String brand = "Honda";
}
class Car extends Vehicle{
    String color = "Blue";
    void start(){
        S.O.P("Car Started");
    }
}
class Bike extends Vehicle{
    int cost = 1,20,000;
    void stop(){
        S.O.P("Bike Stopped");
    }
}
class MainClass{
    psvm(----){
        Car c = new Car();
        Bike b = new Bike();
        S.O.P("Brand:"+c.brand);
        S.O.P("Color:"+c.color);
        c.start();
        S.O.P("**********");
        S.O.P("Brand":"+b.brand);
        S.O.P("Cost:"+b.cost)
        b.stop();
    }
}
```

O/p:-

Honda
Blue
Car Started
****
Honda
1,20,000
Bike Stopped

Method overriding:-

=> The process of inheriting the method and changing the implemen of inherited method called method overriding

* While overriding a method return type, method name, argum should be same.

Note:- While overriding a method, the access modifier should be same or higher vis

While overriding a method, optionally make use of annotation overriding (@override)

Ex:-
```
class Father{
    void Bike(){
        S.O.P("old fashioned father's bike");
    }
}
class Son extends Father{
    @Override
    void bike(){
        S.O.P("new modified son's bike");
    }
    psvm(------){
        Son c = new Son()
        S.bike();
    }
}
```

O/P:- new modified son's bike

Note:- It is always a good practice to use @ override while overri a method @override is an indication to the JVM & programmer that the program is overridden.

Super keyword:-

lower case ← super is a keyword which is used to access super class memb (variables & methods)

Syntax:- Super.variable name;
Super.method name();

```
class Employee{
    void work(){
        S.O.P("Employee is working");
    }
```

```
class Developer extends Employee{
    @Override
    void work(){
        super.work();
        S.O.P("developing APP");
    }
}
class Tester extends Employee{
    @Override
    void work(){
        S.O.P("Testing app");
        super.work();
    }
}
class MainClass{
    psvm(------){
    Developer d = new Developer();
    d.work();
    S.O.P("------");
    Tester t = new Tester();
    t.work();
    }
}
```

O/P:- Employee is working
developing APP

Testing app
Employee is working;

Ex:

```
class Whatsapp1{
    void SendMessage(){
        S.O.P("Single tick supported");
    }
}
class Whatsapp2 extends Whatsapp1{
    @Override
    void SendMessage(){
        S.O.P("double tick supported");
```

```
        super.sendMessage();
    }
    void story(){
        S.O.P("Images are supported");
    }
}
class Whatsapp3 extends Whatsapp2{
    @Override
    void sendMessage(){
        super.sendMessage();
        S.O.P("Blue tick supported");
    }
    @Override
    void story(){
        super.story();
        S.O.P("Videos are supported");
    }
    void payment(){
        S.O.P("can pay using whatsapp");
    }
}
class User{
    psvm(-----){
        Whatsapp3 w3 = new Whatsapp3();
            w3.sendMessage();
            S.O.P(----);
            w3.story();
            S.O.P(----);
            w3.payment();
    }
}
```

O/P:
Single tick supported
double tick supported
blue tick supported

Images are supported
videos are supported

---

can pay using Whatsapp

final:-

* final is a keyword which can be used in variable method and class.
* final variable acts as a constant, whose value cannot be reinitialize.

Ex:- final double PI = 3.14;

* final methods are inherited but can not be overridden.
* final class can't be inherited.

final → variable → cannot re-initialize
final → Method → cannot override
final → class → cannot inherit

Constructor Chaining:-

* The process of one constructor calling another constructor is called as constructor chaining.
* Constructor chaining can be achieved in two ways
  - ↳ In same class using this() [this calling statement].
  - ↳ In diff class using IS-A Relationship [Inheritance] and super() [super calling statement].

Ex:-

```
class Dinga
{
    Dinga(int x){
        this();
        S.O.P(1);
    }
    Dinga(){
        S.O.P(2);
    }
    Dinga(){
        S.O.P(1);
    }
    Dinga(int x){
        this();
        S.O.P(2);
    }
    psvm(----){
        S.O.P("Start");
        new Dinga(50);
        S.O.P("End");
    }
}
```

① O/P:- Start
2
1
End

② Start
1
2
End

② class Student
{
    Student(int age){
        this(5.7);
        S.O.P("Age:"+age);
    }
    Student(String name){
        this(20);
        S.O.P("Name:"+name);
    }
    Student(double height){
        S.O.P("Height:"+height);
    }
    psvm(------){
        new Student("Tom");
    }
}

O/P:-
Height = 5.7
Age = 20
Name : Tom

Note:- * this() calling statement or super() calling statement should be the first executable line within a constructor.

* Recursive chaining is not possible in java. Therefore if there are 'n' constructors we can have only 'n-1' calling statements.

* We cannot have multiple calling statements in a single constructor

Super() [Super Calling Statement]:-
* Super() statement is used to achieve constructor chaining in different classes with the help of IS-A Relationship.
* Super() statement can be categorized into 2 types
-> Implicit super()
-> Explicit super()

Implicit super():-
When we create an object of a class and if that class has a super class and if that super class has a non parameterized constructor, the sub class constructor will call the super class constructor implicitly.

Ex:- class Father{
    Father(){
        S.O.P(1);
    }
}

```
class Son extends Father{
    Son(){
        //implicitly super() is used
        S.O.P(2);
    }
    psvm(------){
        new Son();
    }
}
```

O/P:-
1
2

Explicit super():-

When we create an object of a class and if that class has a super class and if that super class has a parameterized constructor, the sub class constructor will call the super class constructor explicitly. Otherwise we get compile time compilation error.

Ex:-

```
class Father{
    Father(String name){
        S.O.P("Hai"+name);
    }
}
class Son extends Father{
    Son(){
        super("John"); //explicit super()
        S.O.P("Bye");
    }
    psvm(-------){
        new Son();
    }
}
```

O/P:-
Hai John
Bye

Ex:①

```
class A
{
    A(int x)
    {
        this("tom");
        S.O.P(1);
    }
    A(String x)
    {
        S.O.P(2);
    }
}
```

```
class B extends A
{
    B()
    {
        super(10);
        S.O.P(3);
    }
    B(double x)
    {
        this();
        S.O.P(4);
    }
}
class Test
{
    psvm(----)
    {
        new B(5.6);
    }
}
```

O/P:-
2
1
3
4

Note: * Constructor chaining is used to avoid code repetition or code redundancy.
* Constructor chaining will improve code reusability.

Ex:-
```
class Employee
{
    String name;
    int id;
    double salary;
Employee(String name){
    this.name = name;
}
Employee(String name, int id){
    this(name);
    this.id = id;
}
Employee(String name, int id, double salary){
    this(name, id);
    this.salary = salary;
}
psvm(----){
    Employee emp = new Employee("Tom", 100, 45.2);
    S.O.P(emp.name);
    S.O.P(emp.id);
    S.O.P(emp.salary);
}
```

O/P:
Tom
100
45.2

```
class Person{
    String name;
    int age;
    Person(String name){
        this.name = name;
    }
    Person(String name, int age){
        this(name);
        this.age = age;
    }
    p s v m(------){
        Person p = new Person("Tom", 22);
        S.O.P("Name:" + p.name);
        S.O.P("Age:" + p.age);
    }
}
```

O/P:-
Name: Tom
Age: 22

Advantages of Inheritance & Constructor chaining:-

* Code repitition or code redundancy is avoided.
* Code reusability is achieved.

Ex:-

```
class Person{
    int id;
    String name;
    Person(int id, String name){
        this.id = id;
        this.name = name;
    }
}
class Student extends Person{
    int marks;
    Student(int id, String name, int marks){
        super(id, name);
        this.marks = marks;
    }
}
class Employee extends Person{
    double salary;
    Employee(int id, String name, double salary){
```

```
        super(id,name);
        this.salary= salary;
    }
}
class Solution
{
    p s v m(------){
        Student std= new Student(101,"Tom",98);
        S.O.P("Id:"+std.id+"\n Name:"+std.name+"\n Marks:"+std.marks);
        S.O.P("----------");
        Employee emp=new Employee(111,"Jack",49000.20);
        S.O.P("Id:"+emp.id+"\n Name:"+emp.name+"\n Salary:"+emp.salary);
    }
}
```

O/P:-

```
Id:101
Name:Tom
Marks:98
----------
Id:111
Name:Jack
Salary:49000.20
```

Ex:-

```
class Vehicle{
    String brand;
    int cost;
    Vehicle(String brand,int cost){
        this.brand= brand;
        this.cost= cost;
    }
    void display(){
        S.O.P("Brand:"+brand);
        S.O.P("Cost:"+cost);
    }
}
class Car extends Vehicle{
    String fuel;
```

```
    Car(String brand, int cost, String fuel){
        super(brand, cost);
        this.fuel = fuel;
    }
    void display(){
        super.display();
        S.O.P("Fuel:" + fuel);
    }
}
class Bike extends Vehicle {
    String color;
    Bike(String brand, int cost, String color){
        super(brand, cost);
        this.color = color;
    }
    void display(){
        super.display();
        S.O.P("Color:" + color);
    }
}
class MainClass {
    psvm(------){
        Car c = new Car("BMW", 6500000, "Petrol");
        c.display();
        Bike b = new Bike("Honda", 85000, "Black");
        b.display();
    }
}
```

O/P:-

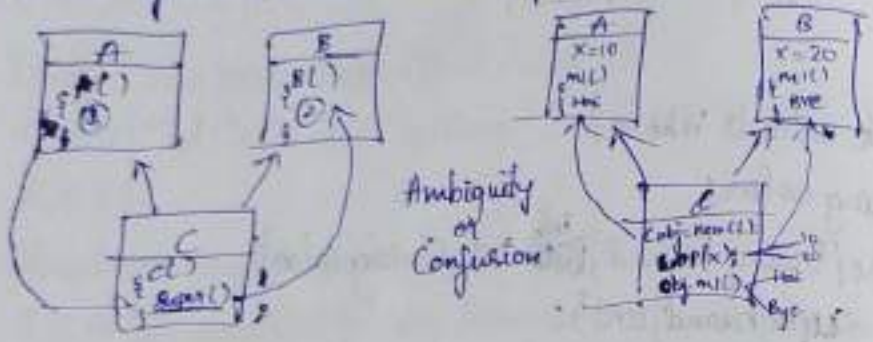
```
Brand: BMW
Cost: 6500000
Fuel: Petrol
-------------
Brand: Honda
Cost: 85000
Color: Black
```

Q) Why Java Does not support multiple inheritance with respect to classes? ★★★

Sol ★ There is a confusion or ambiguity created during Constructor chaining

★ There is an ambiguity created when super class is having same variable names.

★ There is a confusion created when super class have same method names.



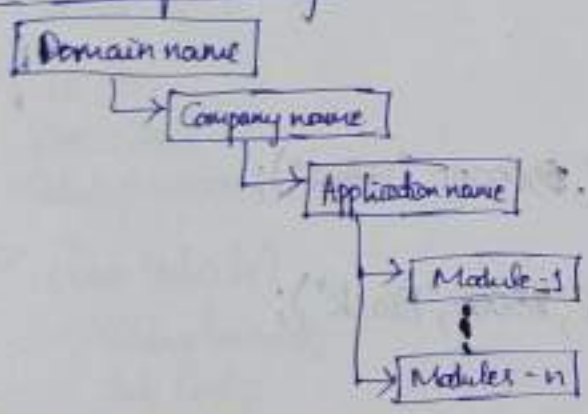
Packages:-

★ Packages is a folder or directory in order to store a set of java program

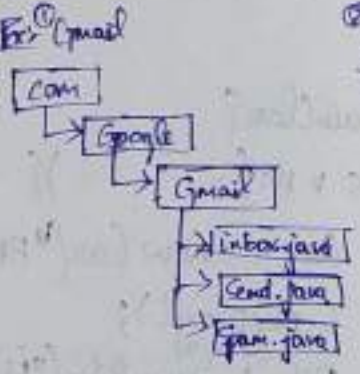
★ Searching becomes easy with the help of packages.

★ There is good maintenance of the programs.

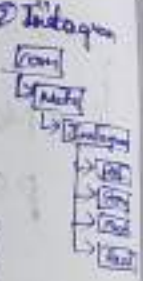
Structure of a Package:-



Ex: ① Gmail



② Instagram



Access Specifier/Access Modifier:-

★ Access Modifiers are used to specify the accessibility of a specific member.

★ In other words, access modifiers are used to provide or set the visibility of a member.

★ The different access modifiers are as follows:-

→ public

→ private

→ default

→ protected

	Outer Class	Inner Class	Variable	Method	Constructor
public	✓	✓	✓	✓	✓
private	✗	✓	✓	✓	✓
default	✓	✓	✓	✓	✓
protected	✗	✓	✓	✓	✓

	Accessing in Same class	Accessing in diff class same package	Accessing in diff class & diff package
public	✓	✓	✓ (import)
private	✓	✗	✗
default	✓	✓	✗
protected	✓	✓	✓ (IS-A + import)

→ Globally (Everywhere)
→ Class level
→ Package Level

Note :- * Public members can be accessed globally (~~everywhere~~ public members are visible everywhere).
* Private members can be accessed only within the same class.
* Default members are accessible only within the same package.
* Protected is similar to default but protected members can be accessed outside the package in case of IS-A Relationship.

Accessing public members within same class :-

Ex.
```
package com;
public class Student {
    public int age=25;
    public Student(){
        S.O.P("In Student");
    }
    public void study(){
        S.O.P("Studying");
    }
    public static void main(String[] args){
        Student s = new Student();
        S.O.P("Age:"+s.age);
        s.Study();
    }
}
```

O/P :-
In Student
Age : 25
Studying

Accessing public members in different class:

Ex:

```
package com;
public class Employee{
    public static int id=101;
    public String name="Tom";
}
package com;
public class Solution{
    public static void main(String[] args){
        S.O.P("Id:"+Employee.id);
        Employee emp=new Employee();
        S.O.P("Name:"+emp.name);
    }
}
```

O/P:
Id : 101
Name : Tom

Accessing public members in different package:

Ex:

```
package com;
public class Car{
    public int cost=500000;
}
package org;
import com.Car;
public class MainClass{
    psvm(------){
        Car c=new Car();
        S.O.P(c.cost);
    }
}
```

O/P:
500000

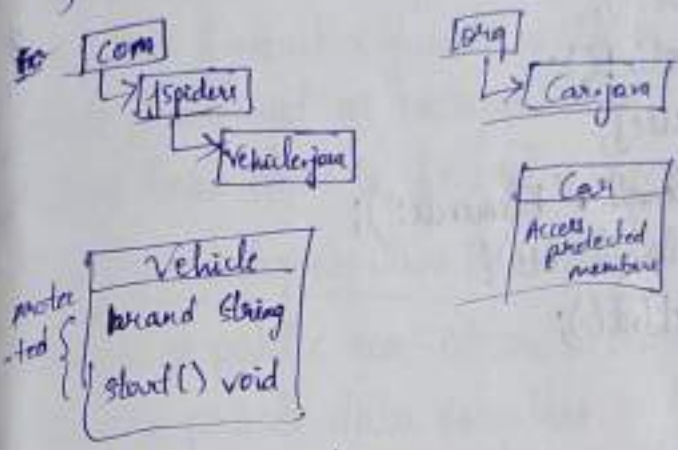
Accessing protected members in different/outside package:

Ex:

```
package com;
public class Father{
    protected String name="Tom";
}
package org;
import com.Father;
```

O/P:
Tom

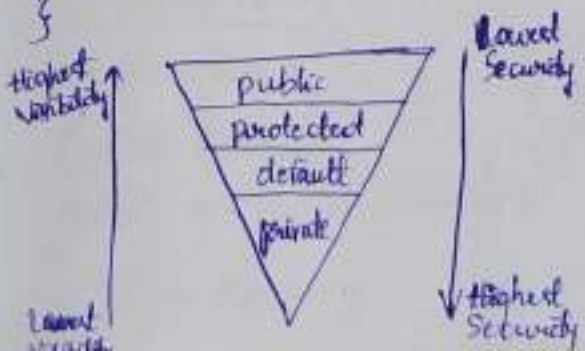
```
public class Son extends Father {
    psvm(------){
        Son s= new Son();
        S.O.P(s.name);
    }
}
```




```
package com.jspiders;
public class Vehicle {
    protected String brand = "BMW";
    protected void start() {
        S.O.P ("BMW Started");
    }
}
package org;
import com.jspiders.Vehicle;
class Car extends Vehicle {
    psvm(------){
        Car c=new Car();
        S.O.P(c.brand);
        c.start();
    }
}
```

O/P:-

BMW
BMW Started



Ex:

```
package org;
import java.util.Scanner;
class Solution{
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        S.O.P("Enter the Size:");
        int size = scan.nextInt();
        int[] a = new Int[size];
        S.O.P("Enter "+a.length+" Elements:");
        for (int i=0; i<a.length; i++){
            a[i] = scan.nextInt();
        }
        scan.close();
        S.O.P("Printing Array Elements:");
        for (int i=0; i<a.length; i++){
            S.O.P(a[i]);
        }
        S.O.P("----------");
        int sum=0;
        for (int i=0; i<a.length; i++){
            sum = sum + a[i];
        }
        S.O.P("SUM:"+sum+"\nAVG:"+(sum/a.length));
    }
}
```

O/P:-

```
Enter the Size:
3
Enter 3 Elements:
10
20
30
Printing Array Elements:
10
20
30
----------
SUM:60
AVG:20
```

Encapsulation :-

* The process of grouping or wrapping data members (variables) and member functions (methods) in a single entity called as class is referred as encapsulation.
* In other words the process of binding & protecting data members & member functions in a single unit called as class is defined as encapsulation.
* Java Bean class is the best example for encapsulation.

Rules for developing Java Bean class:-

1) Create a public non-abstract class.
2) Declare private data members.
3) Declare public getter method and setter method.
4) Declare a public Non-parameterized constructor. } Optional
5) Class should implement Serializable Interface. }

Problem statement:-

```
public class Person                    //Non-abstract class
{                                       ⇓
  private int age;                      (∵ No abstract keyword is
}                                        used before class)
class TestPerson
{
    psvm() {
        Person p = new Person();
        S.O.P(p.age);          //Compile time error
        p.age = 20;            //Compile time error
    }                           ⇓
}                               (∵ of private variable)
```

Correct statements:-

```
public class Person
{
    private int age;
    public void setAge(int age)
    {
        this.age = age;
    }
```

```
    public int getAge()
    {
        return age;
    }
}
public class TestPerson
{
    P S V M ()
    {
        Person p = new Person();
        p.setAge(25);
        int age = p.getAge();
        S.O.P("Age:"+age);
        S.O.P("------");
        S.O.P("Age:"+p.getAge());
    }
}
```

O/P:-

Age:25

Age:25

6) Develop a Java program by following the below scenarios:

a) Create a public Non-abstract class called as Car.

b) Declare a private data member called as brand.

c) Declare public getter method & setter method for the attribute representation.

d) Declare another class called as Solution & under main method, create an instance of car & call this setter & getter method

```
public class Car
{
    private String brand;
    public void setBrand(String brand)
    {
        this.brand = brand;
    }
    public String getBrand()
    {
        return brand;
    }
}
public class Solution
{
```

```
psvm(){
    Car c=new Car();
    ~~c.getBrand("TATA");~~
    c.setBrand("TATA");
    S.O.P("Brand:"+c.getBrand());
  }
}
```

O/P:
Brand: TATA

Program:-

```
public class Employee
{
  private int id;
  private String name;
  public void setId(int id){
    this.id=id;
  }
  public void setName(String name){
    this.name=name;
  }
  public int getId(){
    return id;
  }
  public String getName(){
  { return name;
  }
}
public class TestEmployee{
    psvm(){
      Employee emp=new Employee();
      emp.setId(101);
      emp.setName("Tom");
      S.O.P("Id:"+emp.getId());
      S.O.P("Name:"+emp.getName());
    }
}
```

O/P:-
Id: 101
Name: Tom

Advantages of Java Bean class or Encapsulation:-

1) Readability of the program is improved.
2) Validations can be performed where in we can avoid illogical initialization.
3) We can make the data as read only by having only getter method, Write only by having only setter method, Read & Write by having both getter method & setter method.

Example (for 2nd advantage):-

```
public class Student {
    private int age;
    public void setAge (int age){
        if(age>0)
        {
            this.age = age;
            S.O.P("Age is initialized");
        }
        else{
            S.O.P("Invalid age");
        }
    }
    public int getAge (){
        return age;
    }
}
import java.util.Scanner;
public class TestStudent {
    p s v m ( ){
        Scanner sc = new Scanner(System.in);
        Student s = new Student ();
        S.O.P("Enter Age:");
        int age = sc.nextInt();
        s.setAge(age);
        S.O.P ("Age is" + s.getAge());
    }
}
```

O/P:-

1) Enter Age:
25
Age is initialized
Age is 25

2) Enter Age:
-20
Invalid age

Abstract:-

Abstract is a keyword which can be used with a class and method.

Concrete class:-

* If a class is not declared using abstract keyword, we refer it as concrete class or non-abstract class.
* Concrete class can allow only concrete methods.

Ex:-
```
class Demo{                    //concrete class
    void sleep()               //concrete method
    {
      ==
    }
}
```

Abstract class:-

* If a class is declared using abstract keyword, we refer it as abstract class.
* Abstract class can allow both Abstract method & concrete method

Ex:-
```
abstract class Car             //abstract class
{
  void sleep()                 //concrete method
  {
    ==
  }
  abstract void work();        //Abstract method
}
```

Concrete method:-

If a method has both declaration & implementation, we refer it as Concrete method.

Ex:-
```
void sleep()      //Method declaration
{
  == }            //Method implementation
}
```

Abstract method:-

→ If a method has only declaration & no implementation, we refer it as Abstract method.

* All abstract method should be declared using abstract keyword.

Ex:- abstract void work();

* A class can achieve IS-A Relationship with an abstract class using extends keyword.

*** Contract of Abstract or Rules to follow when a class inherits an abstract class:-

1) When a class inherits an abstract class, we have to override the inherited abstract method.

2) When a class inherits an abstract class & if we do not want to override the inherited abstract method, then make the class as abstract class.

Ex:- 1)
```
abstract class Employee{
    abstract void work();
}
class Developer extends Employee{
    @Override
    void work(){
        S.O.P("Working");
    }
    psvm(){
        Developer d=new Developer();
        d.work();
    }
}
```

o/p:-
Working

2)
```
abstract class Swiggy{
    abstract void orderFood();
}
abstract class Phonepe extends Swiggy{
    abstract void payAmount();
}
class Person extends Phonepe{
    @Override
    void orderFood(){
```

```
    S.O.P("Ordered Food");
  }
  @Override
  void payAmount(){
    S.O.P("Payment Successful");
  }
  psvm(){
    Person p= new Person();
    p.orderFood();
    p.payAmount();
  }
}
```

O/P:-
Ordered Food
Payment Successful

Note:-

* In Java we can create an object of only concrete class, But we cannot create an object of Abstract class.
* Both abstract class & concrete class can have constructor, but the abstract class constructor should be invoke by the subclass constructor either implicitly or explicitly using Super calling statement.

Ex:-

```
abstract class Student {
    Student (String name){
        S.O.P ("Hi"+name);
    }
}
class Tom extends Student {
    Tom(){
        Super("Jack");
        S.O.P("Hi Tom");
    }
    psvm(){
        new Tom();
    }
}
```

O/P:-
Hi Jack
Hi Tom

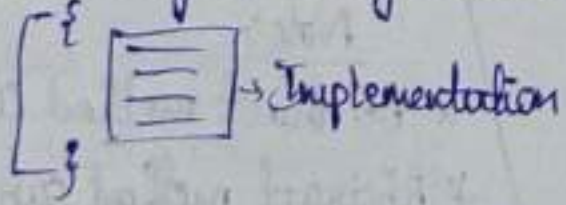
Note:-

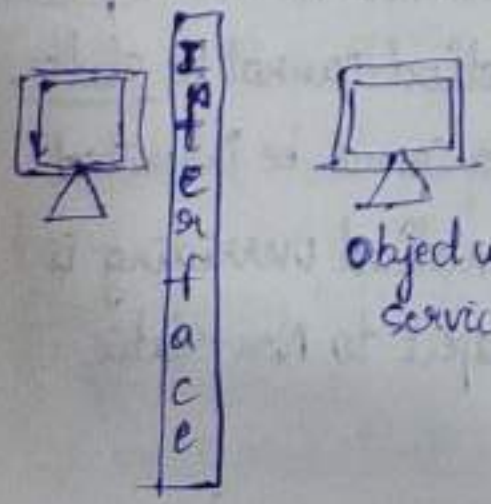
* Abstract method cannot be final
* Abstract method cannot be private
* Abstract method cannot be static.
* Private member cannot be inherited
* Inheritance & method overriding is always with respect to Non-Static members.

Interface:-

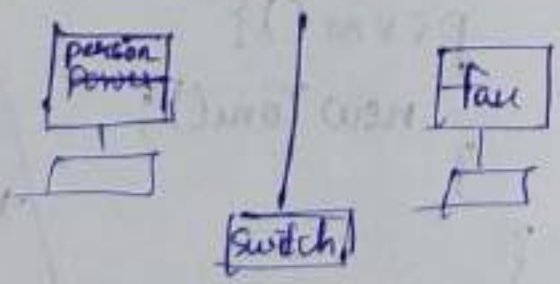
* Interface is a Java type definition which has to be declared using "interface" keyword.
* Interface is also a medium between two ~~software machine~~ systems, one is "client" & another ~~software machine~~ systems is "object with services or resources".

Syntax:- interface InterfaceName





Ex-



* Interface can have variables but they are automatically "public, static & final".
* Interface can allow only abstract methods & they are automatically "public & abstract".
* A class can achieve "IS-A Relationship" with an interface with the help of "implements" keyword.

IS-A	Super class	Super interface
Sub class	extends	implements
Sub interface	X	extends

* When a class implements an interface, we have to follow the <u>contract of abstract</u>.
* When a class implements an interface we have to either override the inherited abstract method or make the class as abstract.
* While overriding a method the access modifier should be same or higher visibility.

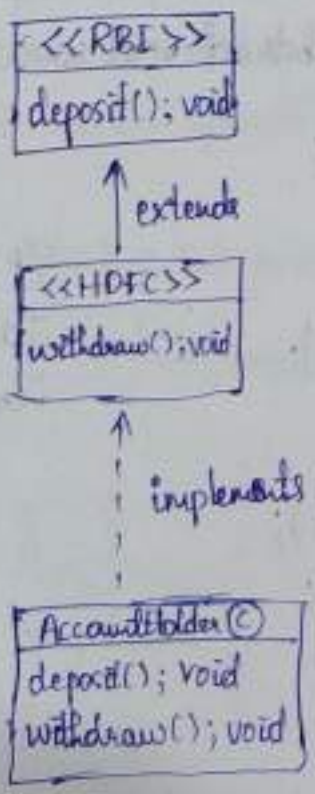
public - public
default - public

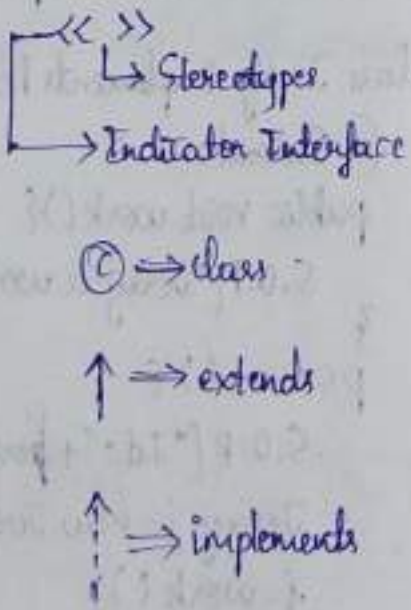
Ex:- 1)
```
interface Person
{
    int id=101;      //public static final int id=101;
    void work();     //public abstract void work();
}
class Jerry implements Person {
    @Override
    public void work(){
        S.O.P("Jerry is working");
    }
    psvm(){
        S.O.P("Id:"+Person.id);
        Jerry j=new Jerry();
        j.work();
    }
}
```

O/P:-
Id : 101
Jerry is working

2)
```
interface RBI
{
    void deposit();
}
interface HDFC extends RBI
{
    void withdraw();
}
class AccountHolder implements HDFC
{
    @Override
    public void deposit() {
        S.O.P("Amount Deposited");
    }
    @Override
    public void withdraw(){
        S.O.P("Amount withdrawn");
    }
    psvm(){
        AccountHolder al=new AccountHolder();
        al.deposit();
        al.withdraw();
    }
}
```

O/P:-
Amount Deposited.
Amount withdrawn





(5) WAJP by following below scenario:-

a) Create a class called as Student

b) Declare 2 attributes called as age & name

c) Initialize those attributes with the help of a constructor.

d) Create another class as Solution & under main method create 2 instances of students & print their details.

(6) WAJP by following below scenarios:-

a) Create a public Non-abstract class called as Employee

b) Declare a private attribute called as name.

c) Declare a respective public getter & setter methods.

d) Create another class as Solution & under main method create an instance of Employee & call the getter & setter method.

---

5) class Student {

```
    int age;
    String name;
    Student (int age, String name){
        this.age = age;
        this.name = name;
    }
}
class Solution{
    psvm ( ){
        Student s1 = new Student(21, "Jerry");
        Student s2 = new Student (22, "Tom");
        S.O.P ("Name: "+s1.name+ "Age: "+s1.age);
        S.O.P ("Name: "+s2.name+ "Age:" +s2.age);
    }
}
```

O/P:-

Name: Jerry Age: 21

Name: Tom Age: 23

6) public class Employee {

```
    private String name;
    public void setName (String name){
        this.name = name;
    }
    public String getName ();
```

```
        return name;
    }
}
class Solution{
    P.S.V.M(){
        Employee e = new Employee();
        e.setName("Tom");
        S.O.P("Name: "+e.getName());
    }
}
```

Name: Tom

Note:-

* A class can implement any number of interfaces [Multiple inheritance with respect of interfaces is possible]
* A class can extend another class & implement any number of interfaces.
* Interface doesnot contain Constructors.
* We can not create an object of interfaces

* Abstract method cannot be private, static & final because the goal of having an abstract method is to override. But if the method is private or static or final we cannot override.

Examples:-

1)
```
class Amazon{
    void work(){
        S.O.P("Working at Amazon");
    }
}
interface Jspiders{
    void developApp();
}
interface Qspiders{
    void testApp();
}
class Employee extends Amazon implements Jspider, Qspiders
{
```

```
@Override
public void developApp(){
    S.O.P("Developing App");
}
@Override ~~void testApp()~~
    S.O.P public void testApp(){
        S.O.P("Testing App");
    }
//optionally it is possible to override work() from amazon class
    psvm(){
        Employee e = new Employee();
        e.work();
        e.developApp();
        e.testApp();
    }
}
```

O/P:-
Working at Amaz
Developing App
Testing App

	Object Creation	Constructor	Variables	Abstract method	Concrete method
Concrete class	✓	✓	✓	✗	✓
Abstract class	✗	✓ [super()]	✓	✓	✓
Interface	✗	✗	✓	✓	No, But from JDK 1.8 default & static Concrete method

```
interface A{
    AM
}
abstract class B{
    AM&CM
}
class C{
    CM
}
```

Ex:-

House — Planning (I), Constructing (A), Built (CC)

TYPE CASTING:-

* The process of converting one type of data into another type of data is called as Type Casting or [Type Conversion].
* Type Casting is divided into:-

1) Primitive Casting — Widening / Narrowing
2) Non-Primitive Casting — Upcasting / Downcasting } ***

I) Widening:-

* The process of converting a smaller type of data into bigger type of data is called as widening.
* Widening will always happen implicitly or automatically.

Ex:- 1) int a = 5;
double b = a;

a	b
5	5.0

2) double i = 50;
S.O.P(i);
o/p:- 50.0

3) char x = 'A';
int y = x;

x	y
'A'	65

∵ ASCII → American Standard Code for Information Interchange

A = 65, B = 66, C = 67, D = 68 . . . . .
a = 97, b = 98, c = 99, d = 100 . . . . .
space ( ) = 32.
Byte → short → int → long → float → double
char

II) Narrowing:-

* The process of converting a bigger datatype into shorter datatype is called as Narrowing.
* Narrowing should always be done explicitly.
* double → float → long → int → short → byte
char

Ex:- 1) double a = 3.15;
int b = (int) a;

a	b
3.15	3

2) int i = (int) 25.6;
S.O.P(i)
o/p:- 25

3) int x = 97
char y = (char) x;

x	y
97	'a'

Upcasting & Downcasting [Non-primitive casting or class type casting]

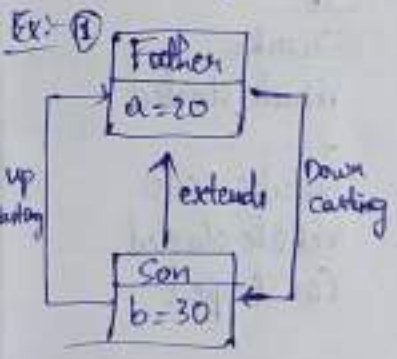
Upcasting:- The process of creating an object of subclass & storing its address into a reference of type Superclass is called as Upcasting.

- Upcasting can also be refered as Superclass reference & subclass object.
- With the upcasted reference or Superclass reference we can access only Superclass members.
- Upcasting will happen implicitly.
- For achieving Upcasting ISA Relationship [Inheritance] is mandatory.

Downcasting:- The process of converting the upcasted reference back to subclass type reference is called as Downcasting.

- With the downcasted reference or subclass reference we can access both superclass members & subclass members.
- Down casting should always be done explicitly.
- For achieving downcasting, upcasting is mandatory.

Ex:- ①

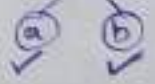


// Upcasting

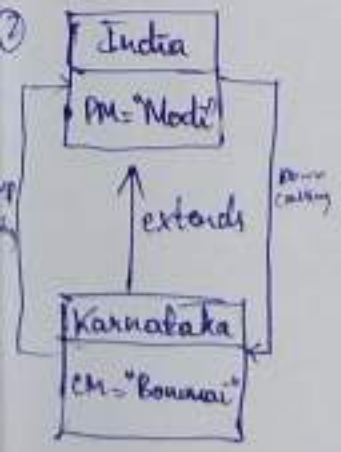
```
Father f = new Son();
```

// Downcasting

```
Son s = (Son) f;
```

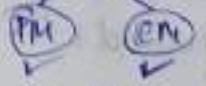


②



(Up) India i = new Karnataka();

(Down) Karnataka k = (Karnataka) i;



India i = k; => This is also a valid upcasting.

Ex-1)

```
class Father {
    int age = 45;
}
class Son extends Father {
    String name = "Tom";
}
class Main class {
    psvm() {
        //Upcasting
        Father f = new Son();
        S.O.P(f.age);
        //Downcasting
        Son s = (Son)f;
        S.O.P(s.age+" "+s.name);
    }
}
```

O/P:-
45
45 Tom

2)

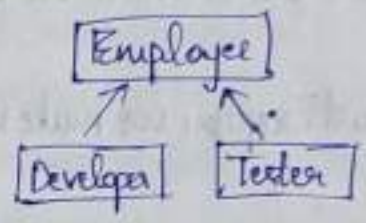
```
class Vehicle {
    String brand = "Suzuki";
    void start() {
        S.O.P("Vehicle Started");
    }
}
class Car extends Vehicle {
    int cost = 5000;
    void stop() {
        S.O.P("Car Stopped");
    }
}
class Test {
    psvm() {
        //Upcasting
        Vehicle v = new Car();
        S.O.P(v.brand);
        v.start();
        S.O.P("-----");
        //Downcasting
        Car c = (Car)v;
        S.O.P(c.brand+" "+c.cost);
        c.start();
        c.stop();
```

O/P:-
Suzuki
Vehicle started

Suzuki 5000
Vehicle started
Car stopped

Specialization & Generalization :-

* If a reference variable is storing only one type of object, we call it as Specialization.
* If a reference variable is storing different types of objects, we call it as Generalization.




```
Developer d = new Developer();
Tester t = new Tester();
```

⇓

Specialization.

```
Employee e = new Developer();
             new Tester();
```

⇓

Generalization

```
void work(Developer d)
{
    ≡
}
void work(Tester t)
{
    ≡
}
```

```
work(new Developer());

work(new Tester());
```

⇓

Specialization

```
void work(Employee e)
{
    ≡
}
```

⟹ Generalization

Class Cast Exception:-

If one object is Upcasted we need to downcast to the same type, otherwise we get ClassCastException.

* In otherwords if one object is Upcasted & downcasted to some other type we get CCE.
* ClassCastException can also occur when we downcast without upcasting.
* In order to avoid ClassCastException we make use of 'instanceof' operation.

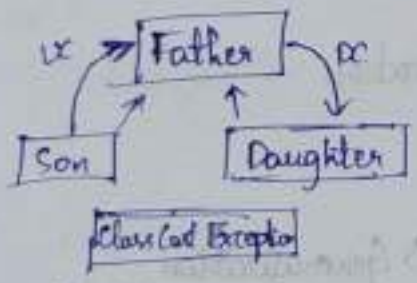
instanceof:-

It is an operator which is used to check if an object is having the properties of class or not.

* In otherwords instanceof is used to check if an object is an instance of a specific class or not.
* The returntype of instanceof operator is boolean.
* Syntax:- object instanceof ClassName

  or

  new ClassName() instanceof ClassName




```
Father f = new Son();
Daughter d = (Daughter)f;
```

instanceof

Ex:-1)

```
class Father {
    int x=10;
}
class Son extends Father {
    int y = 20;
}
```

```
class Daughter extends Father {
    int z=30;
}
class Test {
    psvm() {
        Father f=new Father();
        Son s=new Son();
        Daughter d=new Daughter();
        S.O.P(s instance of Son); //true
        S.O.P(s instance of Father); //true
        S.O.P(d instance of Father); //true
        S.O.P(d instance of Daughter); //true
        S.O.P(f instance of Father); //true
        S.O.P(f instance of Son); //false
        S.O.P(f instance of Daughter); //false
        S.O.P(new Son() instanceof Father); //true
        S.O.P(new Father() instanceof Daughter); //false
    }
}
class Demo {
    psvm() {
        Father obj=new Daughter();
        if(obj instanceof Son) {
            S.O.P("Downcasting to Son");
            Son s=(Son)obj;
            S.O.P(s.x+" "+s.y);
        }
        else if(obj instanceof Daughter)
        {
            S.O.P("Downcasting to Daughter");
            Daughter d=(Daughter)obj;
            S.O.P(d.x+" "+d.z);
        }
    }
}
```

O/P:- Downcasting to Daughter

10 30

OOPS Continuation :- - - - - -

```
class Solution {
    //Father obj = new Son or new Daughter();
    static void display(Father obj){
        if(obj instance of Son){
            Son s=(Son) obj;
            S.O.P(s.x+" "+s.y);
        }
        else if(obj instanceof Daughter){
            Daughter d=(Daughter) obj;
            S.O.P(d.x+" "+d.z);
        }
    }
    psvm(){
        display(new Son());
        display(new Daughter());
    }
}
```

O/P :-

10 20

10 30

Ex2)

```
class Employee {
    String Company="Amazon";
}
class Developer extends Employee {
    void developApp(){
        S.O.P("Developing App");
    }
}
class Tester extends Employee {
    void testApp(){
        S.O.P("Testing App");
    }
}
```

```
Class Runner{
    Employee emp = new Developer(); or new Tester();
    static void work(Employee emp) // Superclass reference
    {
        if(emp instance of Developer){
            Developer d = (Developer)emp;
            S.O.P("Company Name:"+d.company);
            d.developApp();
        }
        else if(emp instance of Tester){
            Tester t = (Tester)emp;
            S.O.P("Company Name:"+t.company);
            t.testApp();
        }
    }
    p s v m ()
    {
        work(new Developer());
        work(new Tester());
    }
}
```

O/P:-

Company Name: Amazon
Developing App
Company Name: Amazon
Testing App

Ex:-3)

```
class India{
    String pm="Modi";
}
class Karnataka extends India{
    String cm="Bommai";
}
class TamilNadu extends India{
    String cm="Stalin";
}
class Goa extends India
{
```

```
    String cm = "Savanth";
}
class Citizen{
    static void selectState(India obj) //Superclass reference
    {
        if(obj instanceof Karnataka){
            S.O.P("State -> Karnataka");
            Karnataka k = (Karnataka)obj;
            S.O.P("PM: "+k.pm+ "\n CM:"+ k.cm);
        }
        else if(obj instanceof TamilNadu){
            S.O.P("State -> TamilNadu");
            TamilNadu t = (TamilNadu) obj;
            S.O.P("PM: "+t.pm+"\n CM: "+t.cm);
        }
        else if(obj instanceof Goa){
            S.O.P("State -> Goa");
            Goa g = (Goa) obj;
            S.O.P("PM: "+g.pm+" \n CM: "+g.cm);
        }
}
psvm(){
    selectState(new Karnataka());
    S.O.P("------");
    selectState(new TamilNadu());
    S.O.P("------");
    selectState(new Goa());
}
}
```

O/P:-

State -> Karnataka
PM: Modi
CM: Bommai.

State -> TamilNadu
PM: Modi
CM: Stalin

State -> Goa
PM: Modi
CM: Savanth

Ex:- 4)

```
class Beverage {
}
class Coffee extends Beverage {
}
class Tea extends Beverage {
}
class Cafe {
    Beverage vendingMachine(int choice)
    {
        if(choice==1){
            return new Coffee();
        }
        else if(choice==2){
            return new Tea();
        }
        else {
            return null;
        }
    }
}
class User {
    psvm () {
        Cafe c=new Cafe();
        Scanner s= new Scanner(System.in);
        S.O.P("Enter choice:");
        S.O.P("1:Coffee \n 2: Tea");
        int choice=s.nextInt();
        Beverage obj= c.vendingMachine(choice);
        if(obj instanceof Coffee){
            S.O.P("Drinking Coffee");
        }
```

```
    else if(obj instanceof Tea){
        S.O.P("Drinking Tea");
    }
    else{
        S.O.P("Invalid");
    }
  }
}
```

O/P:-
Enter Choice:
1: Coffee
2: Tea

2	5	1
Drinking Tea	Invalid	Drinking Coffee

POLYMORPHISM:- [Many forms]

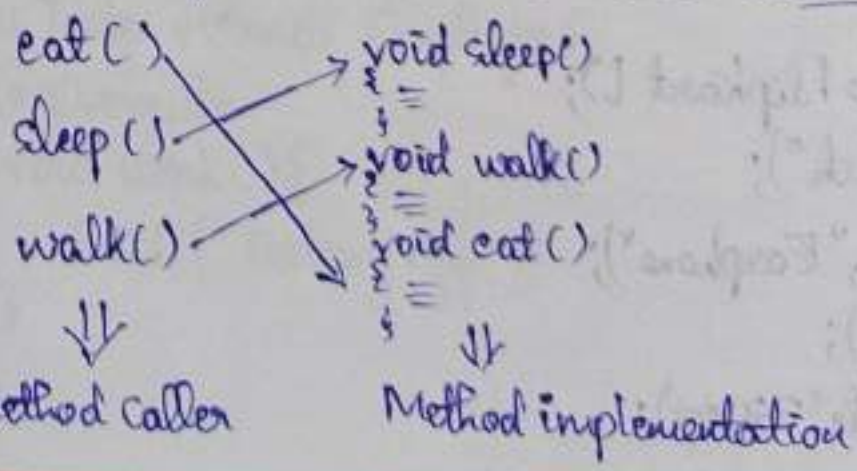
The ability of a method to behave differently when different objects are acting upon it is called as Polymorphism.

* In other words the smartness of a method to exhibit different forms when different objects are acting upon it is called as Polymorphism.
* Polymorphism can be categorized into 2 types:-

1) Compile time polymorphism (CTP)
2) Runtime polymorphism (RTP)

Method Binding:-

The process of mapping or associating the method caller to its method implementation is called as Method Binding.



① Compile Time Polymorphism (CTP):-

* It is achieved with the help of method overloading.
* CTP is also called as Static binding or Early binding.
* During CTP method binding happens during compile time & it is decided by the compiler.
* Out of so many overloaded methods, which method implementation should be executed is decided by compiler during compile time.

Ex:-

```
class Flipkart {
    void purchase(int cost){
        S.O.P("Cost:"+cost);
    }
    void purchase(String product){
        S.O.P("Product:"+product);
    }
    void purchase(String product, int cost){
        S.O.P("Product:"+product+"Cost:"+cost);
    }
    void purchase(int cost, String product){
        S.O.P("Cost:"+cost+"Product:"+product);
    }
}
class Customer {
    psvm(){
    Flipkart f = new Flipkart();
    f.purchase("Watch");
    f.purchase(5000,"Earphone");
    f.purchase(2000);
    f.purchase("Mobile",25000);
    }
}
```

(2) Run Time Polymorphism:-

* RTP is achieved with the help of
  i) IS-A Relationship (Inheritance)
  ii) Method Overriding
  iii) Upcasting
* RTP is also called as Dynamic Binding or Late Binding or Dynamic Method Dispatch
* During RTP method binding happens during runtime & JVM decides

---

Note:-
* If a method is overridden we always get the overridden implementation as the output even if it is upcasted reference or downcasted reference.

Ex:-
1)
```
class Employee {
    void work() {
        S.O.P ("Working");
    }
}
class Developer extends Employee {
    @Override
    void work() {
        S.O.P ("Developing App");
    }
}
class Tester extends Employee {
    @Override
    void work() {
        S.O.P ("Testing App");
    }
}
```

```
class Runner{
    Employee emp=new Developer(); or new Tester();
    static void display(Employee emp){
        emp.work();
    }
    psvm(){
        display(new Developer());
        display(new Tester());
        S.O.P("------");
        Employee e1=new Developer();
        e1.work();
        S.O.P("------");
        Employee emp; //SuperclassReference
        emp=new Developer();
        emp.work();
        emp=new Tester();
        emp.work();
    }
}
```

Employee e2=new Tester();
e2.work();

O/P:-

```
Developing App
Testing App
------
Developing App
Testing App
------
Developing App
Testing App
```

Ex 2)

```
class FoodDeliveryApp{
    void order(){
        S.O.P("Ordering Food");
    }
}
class Swiggy extends FoodDeliveryApp{
    @Override
    void order(){
        S.O.P("Ordering Biryani");
    }
}
```

```
class Zomato extends FoodDeliveryApp {
    @Override
    void order(){
        S.O.P("Ordering Curd Rice");
    }
}
class Customer {
    static void selectApp(FoodDeliveryApp app){
        app.order();
    }
    psvm() {
        selectApp(new Swiggy());
        selectApp(new Zomato());
        S.O.P("------");
        FoodDeliveryApp f1 = new Swiggy();
        f1.order();
        FoodDeliveryApp f2 = new Zomato();
        f2.order();
        S.O.P("------");
        FoodDeliveryApp app; // Superclass Reference
        app = new Swiggy();
        app.order();
        app = new Zomato();
        app.order();
    }
}
```

O/P:- Ordering Biryani
Ordering Curd Rice

Ordering Biryani
Ordering Curd Rice

Ordering Biryani
Ordering Curd Rice

Polymorphism

- CTP
  - ↓
  - MOL (Method Overloading)
  - ↓
  - Method Binding happen during CT
    Compiler decides
  - ↓
  - Static Binding or Early Binding

- RTP
  - ↓
  - IS-A
    MOR (Method Overriding)
    UC (Upcasting)
  - ↓
  - Method Binding happen during RT
    JVM decides
  - ↓
  - Dynamic Binding or Late Binding

Singleton class / Singleton Design Pattern:-

The process of creating only one object or only one ~~interpa~~ instance for a class is called as singleton design pattern.

Rules for creating singleton class:-

1) Declare a private constructor
   [Object cannot be created outside the class, we can create only within the class].
2) Declare a public static Helper method which creates one object and return the object.
3) Declare a private static Non-primitive reference variable.

Ex:- 1)

```
class Account {
    private static Account obj; //default value = null
    private Account () {
        S.O.P("Object created");
    }
    public static void createObject () {
        if(obj == null){
            obj = new Account ();
        }
        else {
            S.O.P ("Cannot create object");
        }
    }
}
class Test {
    p s v m ( ) {
        Account . createObject ();
        Account . create Object ();
        Account . create Object ();
    }
}
```

O/P:-

Object created
Cannot create Obje
Cannot create Obje

Ex:- 2)

```
class PrimeMinister {
    String name = "Modi";
    private static PrimeMinister pm;
    private PrimeMinister () {
        S.O.P("PM Elected");
    }
    public static PrimeMinister getInstance () {
        if(pm == null)
        {
```

```
        pm = new PrimeMinister();
    }
    return pm; //return new PrimeMinister();
    }
}class Citizen{
    p s v m ( ){
        //Internally PrimeMinister pm = new PrimeMinister();
        PrimeMinister pm = PrimeMinister.getInstance();
        S.O.P("Name:"+pm.name);
    }
}
```

O/P:- PM Elected
Name: Modi

Ex:- 3)

```
class AadhaarCard{
    String name = "Tom";
    int aadhaarNo = 123;
    private static AadhaarCard ac;
    private AadhaarCard(){
        S.O.P("Aadhaar Card generated");
    }
    public static AadhaarCard getInstance(){
        if(ac==null){
            ac = new AadhaarCard();
        }
        return ac; //return new AadhaarCard();
    }
}class Person{
    p s v m ( ){
        //Internally AadhaarCard ac = new AadhaarCard();
```

```
    AadhaarCard ac = AadhaarCard.getInstance();
    S.O.P("Name:"+ac.name);
    S.O.P("AadhaarNo:"+ac.aadhaarNo);
  }
}
```

O/P:- Aadhaar card generated
Name : Tom
AadhaarNo : 123

Abstraction:-

* The process of hiding the unnecessary implementation details & showing only the essential functionality to the user with the help of an abstract class or interface is called as Abstraction.
* In other words hiding the implementation & showing the functionality to the user is called as Abstraction.
* In order to achieve abstraction we have to follow below rules.
  i) Abstract class (or) interface with abstract method
  ii) IS-A relationship (Inheritance)
  iii) Method overriding
  iv) Upcasting

Ex Program:-

1)
```
package com;
public abstract class Person{
    abstract void work();
}
class Employee extends Person {
    @Override
    void work()
    {
```

```
    S.O.P("Working");
  }
}
class Test{
    psvm(){
        Employee e = new Employee();
        e.work(); //normal invoke of overridden method
        Person p = new Employee(); //Upcasting
        p.work(); //Abstraction
    }
}
```

o/p:
Working
Working

Ex:2)
```
package bankapp;
public interface Bank{
    void deposit(int amount);
    void withdraw(int amount);
    void checkBalance();
}
public class ATM implements Bank{
    int balance = 1000;
    @Override
    public void deposit(int amount){
        S.O.P("Depositing Rs." + amount);
        balance = balance + amount;
        S.O.P("Amount deposited Successfully");
    }
    @Override
    public void withdraw(int amount){
        S.O.P("Withdrawing Rs." + amount);
```

```
    balance = balance - amount;  //balance -= amount
    S.O.P("Amount withdrawn successfully");
  }
  @Override
  public ~~class~~ void checkBalance(){
      S.O.P("Available Balance: Rs."+balance);
  }
}
public class AccountHolder{
    psvm() {
    //ATM obj = new ATM();  //Normal invoking
    Bank obj = new ATM();  //Upcasting
    obj.checkBalance();
    S.O.P("-------");
    obj.deposit(4000);
    obj.checkBalance();
    S.O.P("-------");
    obj.withdraw(9000);
    obj.checkBalance();
    }
}
```

O/P:-

```
Available Balance: Rs.1000
---------
Depositing Rs.4000
Amount Deposited Successfully
Available Balance: Rs.14000
---------
Withdrawing Rs.9000
Amount Withdrawn Successfully
Available Balance: Rs.5000
```

(Q)

```
public class Solution{
    psvm() {
        S.O.P("Welcome to ATM");
        S.O.P("----------");
        Scanner s = new Scanner(System.in);
        Bank b = new ATM();
```

```
while(true){
    S.O.P("1.Deposit 2.Withdraw 3.check Balance
    4.Exit");
    S.O.P(" Enter Choice: ");
    int choice = s.nextInt();
    switch(choice){
        case 1 : S.O.P("Enter amount to be deposited");
                 int dAmt = s.nextInt();
                 b.deposit(dAmt);
                 break;
        case 2 : S.O.P("Enter Amount to be Withdraw");
                 int wAmt =
                 b.withdraw(s.nextInt());
                 /* int wAmt = s.nextInt();
                  b.withdraw(wAmt);*/
                 break;
        case 3 : b.checkBalance();
                 break;
        case 4 : S.O.P("Thank You!!");
                 System.exit(0);
        default : S.O.P("Invalid choice");
    }
}
}
}
```

o/p:-

```
Welcome to ATM
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.check Balance 4.Exit
Enter choice:
3
Available Balance: Rs.10000
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.Check Balance 4.Exit
Enter choice:
2
Enter Amount to be Withdrawn
6000
Withdrawing Rs.6000
Amount Withdrawn Successfully
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.check Balance 4.Exit
Enter choice:
3
Available Balance: Rs.4000
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.check Balance 4.Exit
Enter choice:
1
Enter Amount to be Deposited
5000
Depositing Rs.5000
Amount Deposited Successfully
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.Check Balance 4.Exit
Enter choice:
3
Available Balance: Rs.9000
- - - - - - - - - - -
1.Deposit 2.Withdraw 3.Check Balance 4.Exit
Enter choice:
546
Invalid choice
```

```
1.Deposit 2.Withdraw 3.Check Balance
4.Exit
Enter choice:
4
Thank You!!
```

Java Libraries:-

* It is a collection of predefined packages.
* Each package is a collection of classes and interfaces.
* Each class or interface is a collection of variables & methods.
* All the predefined packages are present in a zip file called as src.zip (source) or a jar file called as rt.jar (runtime)
* A few predefined packages are as follows:-

→ java.lang (most important)
→ java.util
→ java.io
→ java.sql
→ java.awt
→ java.applet
→ java.net

A few Important classes in the below packages

java.lang	java.util	java.io
Object	Scanner	File
String	Arraylist	File Reader
Thread	Hash Set	File Writer
System	⋮	⋮
⋮		

java.lang package

java.lang package is automatically imported in all classes and interfaces.

Object :-

* Object is a predefined class present in java.lang package.
* Object is referred as the supermost class in java because every class in java (predefined & user defined) will implicitly inherit object class.

Ex:-
```
class Car extends Object
{
    Object obj = new Car();   UP-casting
}
```

A few Important methods present in object class

1) public String toString()
2) public int hashCode()
3) public boolean equals (Object obj)
4) public void notify()
5) public void notifyAll()
6) public void wait()
7) public void wait (long a)
8) public void wait (long a, int b)
9) public Class getClass()
10) protected void finalize()
11) public protected Object clone()

① toString():- * toString() is used to return the string representation of an object.

* When we print the reference variable implicitly, toString() gets called
* toString() will return the string representation of the object in the below format.

⇒ FullyQualifiedClassName@HexadecimalValue of the HashCode.

Note:-

* FullyQualifiedClassName means considering the ClassName from the package level.

Syntax of toString():-

```
public String toString(){
}
```

Program without overriding toString():-

```
class Car{
    public static void main(String[] args){
        Car c = new Car();
        // Implicitly calling toString()
        S.O.P(c);
        // Explicitly calling toString()
        S.O.P(c.toString());
    }
}
```

o/p:-

com.Car@33909752
com.Car@33909752

Program after Overriding toString():

```
class Car{
  @Override
  public String toString(){
     return "Car@100";
  }
  psvm(){
     Car c = new Car();
     S.O.P(c);
     S.O.P(c.toString());
  }
}
```

O/P:-
Car@100
Car@100

Ex

```
class Student{
  int age;
  Student(int age){
     this.age = age;
  }
  @Override
  public String toString(){
     return "Age:"+age;
  }
  psvm(){
     Student s = new Student(25);
     S.O.P(s);
     S.O.P(s.toString());
  }
}
```

O/P:-
Age:25
Age:25

Q) WAP by following below scenarios
a) Create a class called as Employee
b) Declare 2 attributes called as id & name

c) Initialize those 2 attributes with the help of a constructor
d) Override toString() by returning the name & id in the below format
(Emp Id of Tom is 101)
e) Create another class called as Solution & under main method create 2 instances of employee & invoke toString() implicitly.

Sol:
```
class Employee{
    int id;
    String name;
    Employee (int id, String name){
        this.id=id;
        this.name=name;
    }
    @Override
    public String toString(){
        return "Emp Id of "+name+" is "+id;
    }
}
class Solution{
    psvm(){
        Employee e1=new Employee(101,"Tom");
        Employee e2=new Employee(102,"Jack");
        S.O.P(e1);
        S.O.P(e2);
    }
}
```

O/P:-

Emp Id of Tom is 101
Emp Id of Jack is 102

~~Overrid~~

Overriding toString() by Storing objects in an Array :-

```
class Car{
    String brand;
    Car(String brand){
        this.brand = brand;
    }
    @Override
    public String toString(){
        return "Brand:"+brand;
    }
}
class TestCar{
    psvm ( ){
        Car c1 = new Car("Suzuki");
        Car c2 = new Car("Hyundai");
        Car c3 = new Car("Audi");
        Car[] c = {c1,c2,c3};
        for(int i=0; i<c.length; i++){
            S.O.P(c[i]);
        }
    }
}
```

O/P:- Brand : Suzuki
⋮ ⋮ ⋮

Q) WAJP to store 3 hotel objects inside an Array & traverse the objects using forloop after overriding toString()

* Hotel Object has 2 attributes Name & Ratings.

```
class Hotel{
  String name;
  double ratings;
  Hotel(String name, double ratings){
    this.name=name;
    this.name= ratings;
  }
  @Override
  public String toString(){
    return "The Hotel "+name+" has ratings of "+ratings;
  }
  public static void main(String[] args){
    Hotel h1= new Hotel("Taj",4.8);
    Hotel h2= new Hotel("Sagar",4.5);
    Hotel h3= new Hotel("Mayura",4.2);
    Hotel [] h={h1,h2,h3};
    for(int i=0; i<h.length; i++)
    {
      S.O.P(c[i]);
    }
  }
}
```

O/P:- The Hotel Taj has ratings of 4.8
The Hotel Sagar has ratings of 4.5
The Hotel Mayura has ratings of 4.2

HashCode Method :-

* HashCode method is used to return a unique id or number to identify an object.
* In other words hash code method is used to return a unique random number which cannot be duplicated.

Syntax:-

```
public int hashCode [redacted] (){
    ----
}
```

Program without overriding hashCode Method

```
package org;
class Student{
    public static void main(String[] args){
        Student s= new Student();
        S.O.P(s.hashCode());
    }
}
```

O/P :-
865113938

Program after overriding hashCode method :-

```
package org;
class Student{
    @Override
    public int hashCode(){
        return 234;
    }
    psvm (){
        Student s= new Student();
        S.O.P(s.hashCode());
    }
}
```

O/P :-
234

③ equals() :-

* equals() is generally used for content comparison
* By default equals() will compare the address or hashCode of two objects.
* In order to compare the content or attributes of two objects, we have to override equals().

Rules for overriding equals method:-

→ Upcasting
→ Downcasting
→ Comparison logic

Syntax:-

```
public boolean equals(Object obj)
```

Program without overriding equals method:-

```
package org;
class Student {
  int marks;
  Student (int marks){
    this.marks=marks;
  }
  public static void main (String[] args){
    Student s1= new Student(50);
    Student s2= new Student(50);
    S.O.P(s1==s2);
    S.O.P(s1.equals(s2));
  }
}
```

O/P:-
false
false

```
package org;
class Student{
  int marks;                                  //s1 → this
                                              //s2 → Obj → s
  Student (int marks){
    this.marks = marks;
  }
  @Override          //Object obj = new Student (50);
   public boolean equals(Object obj){
      Student s=(Student)obj;
      return this.marks == s.marks; //s1.marks=s2.marks
   }
   psvm (){
     Student s1 = new Student (50);
     Student s2 = new Student (50);
     S.O.P (s1.equals (s2)); //s1.equals(new Student(50))
   }
}
```

Op:- true

Write a java program to compare the id and salary of 2 employee objects after overriding equals method.

Sol

```
package org;
class Employee{
  int id;
  int salary;
 Employee (int id, int salary){
   this.id = id;
```

```
    this.salary=salary;
  }
  @override
  public boolean equals(Object obj){
      Employee e=(Employee)obj;
      return this.id==e.id && this.salary==e.salary;
  }
  psvm()
  {
      Employee e1=new Employee(101,50000);
      Employee e2=new Employee(102,50000);
      S.O.P(e1.equals(e2));
  }
}
```

O/P:- false
false

Note:-

In order to compare string objects we make use of equals(), not double equals (==).

Ex:-

```
package org;
class Person{
  int age;
  String name;
  Person(int age, String name){
    this.age=age;
    this.name=name;
  }
  @override
```

```
public boolean equals (Object obj){
    Person p=(Person) obj;
    return this.age==p.age && this.name equals(p.name);
  }
}
```

~~psvm ()~~

~~Person p1=new Person(20, "Tom");~~

~~Person p2=new Person(20, "Tom");~~

~~S.O.P(p1.equals(new Person(20, "Tom")));~~

~~S.O.P(p1.equals~~

```
class Solution{
  psvm ()
  {
    Person p1=new Person(20,"Tom");
    Person p2=new Person(20,"Tom");
    S.O.P(p1.equals(new Person(20,"Tom")));  //true
    S.O.P(p1.equals(p2));//true
    if(p1.equals(p2)){
        S.O.P("Age & name are same");
    }
    else
    {
        S.O.P("Age & name are different");
    }
  }
}
```

O/P:-

true

true

Age and name are same

Method	Access Specifier	Return type	Arguments	
toString()	public	String	X	→ String representation of the object
hashCode()	public	int	X	→ Unique id/no for an object
equals()	public	boolean	(Object)	→ Content Comparison

Assignment:-

1) Write a java program to store 5 laptop objects in an array and traverse the objects using for loop after overriding toString methods
   * note, every laptop will have 3 attributes called as brand, model no, cost.
2) Write a java program to compare 2 mobile objects after overriding equals method where in the attributes are brand & cost.

String:-

* String is a predefined <u>final class</u> present in java.lang package
* string is a class and a datatype, therefore we refer string as a non primitive datatype.
* Default value for any class type or string is null.
* String does not have any fixed size.
* String can also be referred as a set of characters which should be enclosed within double quotes.
* String objects are <u>immutable</u> in nature.
* String objects can be created in two ways.

→ without using new operator (literal)

```
String s="Java";
```

→ using new operator

```
String s=new String("Java");
```

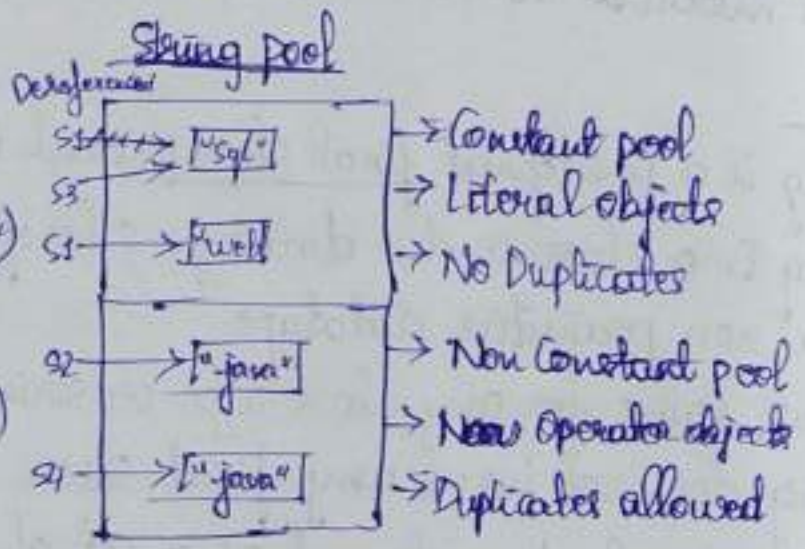
* String objects will get stored inside a memory location called <u>String pool</u>.

* String pool is further divided into
  → Constant Pool
  → Non Constant Pool

* String objects which are created without new operator (literal) are stored inside constant pool and constant pool does not allow duplicates

* String objects created using new operator are stored inside non-constant pool and non constant pool allows duplicates.

① String s1 = "sql".
② String s2 = new String("java")
③ String s3 = "sql"
④ String s4 = new String("java")



<u>Note:-</u>

* String was introduced from jdk 1.0
* String class implements the following interfaces
  → java.io.<u>Serializable</u>, <u>Comparable<String></u>, <u>CharSequence</u>
* String class also implicitly inherits object class and has automatically overridden three methods :-
  → toString()
  → hashCode()
  → equals()

* toString method will generally return the string representation of an object, but in string class it is overridden to return the set of characters.
* hashCode method will generally return a unique id or number for an object but, in string class it is overridden to return ASCII value of the object.
* equals method will generally compare the address or references of 2 objects but in string class equals method is overridden to compare the content of 2 string objects.

<u>Difference b/w double equals (==) & equals () with respect to string</u>

* double equals (==) will always compare the addresses or references of 2 string objects.
* equals() will compare the content of 2 string objects.

Ex:

```
package test;
class Student {
  psvm() {
    Student s = new Student();
    S.O.P(s);
    S.O.P(s.toString());
    S.O.P("-------");
    S.O.P(s.hashCode());
    S.O.P("-------");
    Student s1 = new Student();
    Student s2 = new Student();
    S.O.P(s1.equals(s2));
  }
}
```

O/P:-

```
test.Student@3909752
test.Student@3909752
-------
865113938
-------
false
```

Ex:② class Demo

```
{
  psvm()
  {
    String s=new String("c");
    S.O.P(s);
    S.O.P(s.toString());
    S.O.P("------");
    S.O.P(s.hashCode());
    S.O.P("------");
    String s1=new String("java");
    String s2=new String("java");
    S.O.P(s1==s2);
    S.O.P(s1.equals(s2));
  }
}
```

O/P:-

```
c
c
-------
67
-------
false
true
```

How String objects are immutable in nature or Explain String immutability concept?

(Sol) 1) When we reinitialize a string object, the existing object is not modified rather than that a new object gets created

2) The referenced variable pointing to the old object gets dereferenced and it starts pointing to the newly created object 3) This is how the string objects are immutable in nature

The mutable version of String are as follows:-

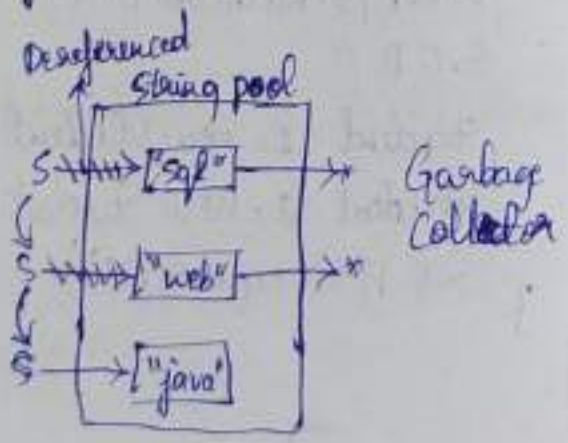
a) String Buffer

b) String Builder

String s="sql"

s="web"

s="java"



String Buffer:-

* String Buffer is a predefined final class present in java.lang package
* String-Buffer was introduced from jdk 1.0.
* String-Buffer is mutable in nature.
* String Buffer implements 2 interfaces
  → Serializable
  → CharSequence

String Builder:-

* String Builder is a predefined final class present in java.lang package
* String Builder was introduced from jdk 1.5.
* StringBuilder is mutable in nature.
* StringBuilder implements 2 interfaces
  → Serializable
  → CharSequence

Methods present in String class

Ex:

```
package com;
public class Demo{
    p s v m ( ){
        String s="Software Developer";
        S.O.P(s.length());  //18
        S.O.P(s.toUpperCase());// SOFTWARE DEVELOPER
        S.O.P(s.toLowerCase());//software developer
        S.O.P(s.startswith("soft"));//false
        S.O.P(s.endswith("per"));//true
        S.O.P(s.charAt(1));//o
        S.O.P(s.indexOf('e'));//7
```

```
        S.O.P(s.contains("ware")); //true
        S.O.P(s.concat("in Amazon")); //Software Developer in Amazon

        String a="java";
        String b="java";
        String c="JavA";
        S.O.P(a.equals(b)); //true
        S.O.P(a.equals(c)); //false
        S.O.P(a.equalsIgnoreCase(c)); //true
    }
}
```

Ex:

```
package com;
class Test{
    p s v m ( ) {
        String s1= new String("Java");
        S.O.P(s1); //Java
        s1.concat("Program");
        S.O.P(s1); // Java
        S.O.P("------"); //------
        String s2 = new String("Java");
        S.O.P(s2); //Java
        s2 = s2.concat("Program");
        S.O.P(s2); //Java Program
        S.O.P("------"); //------
        StringBuffer s3= new StringBuffer("Good");
        S.O.P(s3); // Good
        s3.append(" Morning"); // Good Morning
        S.O.P(s3);
        S.O.P("------") //------
```

```
StringBuilder S4 = new StringBuilder("Chicken");
S.O.P(S4); // Chicken
S4.~~concat~~ append(" Biryani");
S.O.P(S4); // Chicken Biryani
S.O.P("------") //------
String S5 = new String("Dabba");
S.O.P(S5); // Dabba
S5 = S5.concat(" Fellows");
S.O.P(S5); // Dabba Fellows
}
}
```

Difference between String, StringBuffer, StringBuilder

String	StringBuffer	StringBuilder
① It is predefined final class in java.lang package.	① It is a predefined final class in java.lang package.	① It is a predefined final class in java.lang package.
② Introduced in JDK 1.0	② Introduced in JDK 1.0	② Introduced in JDK 1.5
③ Immutable in nature	③ Mutable in nature	③ Mutable in nature.
④ Thread safe (single threaded)	④ Thread safe (single threaded)	④ Not Thread safe [XTS] (Multi Threaded)
⑤ toString(), hashCode(), & equals() are overridden	⑤ Only toString() overridden	⑤ Only toString() overridden
⑥ String objects can be created with or without new operator	⑥ StringBuffer objects can be created only using new operator	⑥ StringBuilder objects can be created only using new operator.
⑦ '+' operator can be used for concatenation	⑦ '+' operator cannot be used for concatenation	⑦ '+' operator cannot be used for concatenation

```
→String s="JAVA";
→Char[] ch = s.toCharArray();
for(int i=0; i<ch.length; i++)
{
    S.O.P(ch[i]);
}
```

s → ["JAVA"]

ch → [J | A | V | A]
0 1 2 3

8) Write a java program to reverse a string

Sol
```
class Demo
{
    public static void main(String[] args)
    {
        String s="JAVA";
        Char[]ch=s.toCharArray();
        ~~for(int i=0; i>ch.length; i++)~~
        ~~{~~
        ~~S.O.P(ch[i]);~~
        ~~}~~
        for(int i=ch.length-1; i>=0; i--)
        {
            S.O.P(ch[i]); //System.out.print(ch[i]); //o/p:- AVAJ
        }
    }
}
```

o/p:-
A
V
A
J

9) Write a java program to revers a string without using toCharArray()

Sol
```
class Demo
{
    public static void main(String[] args)
    {
        ~~String s="JAVA";~~
```

```
String x= "ABCD";
for(int i=s.length()-1; i>=0; i--)
{
    S.O.P(x.charAt[i]);
}
}
}
```

System.out.println
O/P:-
D
C
B
A

System.out.print
O/P:-
DCBA

Q) Write a java program to check if a string is palindrome or not.

```
package org;
public class Palindrome
{
    psvm ()
    {
        String s= "Gadag";
        String reverse = " ";
        for(int i=s.length()-1; i>=0; i--)
        {
            reverse=reverse(s.charAt(i));
        }
        if(reverse.equals(s))
        {
            S.O.P(s+" is palindrome");
        }
        else
        {
            S.O.P(s+" is not palindrome");
        }
    }
}
```

O/P:
Gadag is palindrome

Exception Handling:-

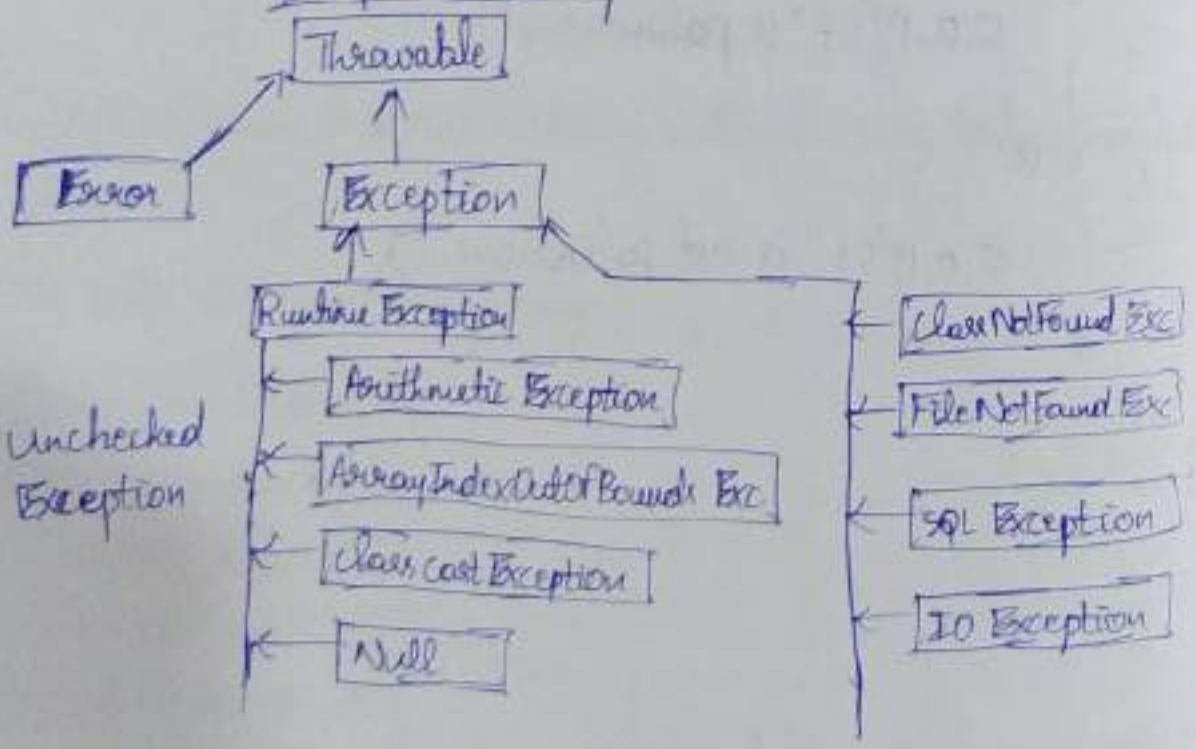
Error - Error is a mistake or a problem which occurs during the execution of program.

* Errors can occur during compile time due to syntax mistake.
* Errors can also occur during runtime generally when we execute a class without main method.
* Errors should always be debugged.

Exception:- Exception is an event or interruption which stops the execution of a program where in the below lines of code will not get executed.

* Exception should always been handled, the process of handling an exception is known as exception handling.
* In other words exception is an runtime interruption which stops the execution of a program, which needs to be handled.

Exception Hierarchy



Note:- All the exceptions are either pre-defined class or user-defined class.

Try and Catch Block:-

When an exception occurs, we generally handle an exception with the help of "try and catch block".

* The critical lines of code or problematic lines of code which might give an exception should be written inside "try block".
* The solution for the exception occured should be written inside catch block.
* catch () block gets executed only when exception occurs.
* try block and catch block should always be defined together.

Syntax:-

```
try{

}
catch (ExceptionName referencevariable){

}
```

①
```
package com;
import java.util.Scanner;
class Demo{
    psvm(){
        S.O.P("Start");
        Scanner s= new Scanner(System.in);
        S.O.P("Enter first no:");
        int x= s.nextInt();
```

```
        S.O.P("Enter second no:");
        int y = S.nextInt();
        scan.close();
        try{
            S.O.P(x/y);
        }
        catch(ArithmeticException e){
            S.O.P("Please do not divide by 0");
        }
        S.O.P("Program ended");
    }
}
```

O/P:- Start

```
Enter First no:
10
Enter Second no:
0
Please do not divide by 0
Program ended.
```

Note 1:- It is always important or necessary to handle an exception using suitable try and catch block.

Note 2:- We can handle the exception either using same type or superclass type.

Internal working when we try to access an invalid index position

* When we access an invalid index position internally an object of arrayIndexOutOfBound exception is thrown.

* The object thrown could be caught either using same class type or super class type.

```
package com;
class Runner{
    psvm(){
        int[] a = {10,20,30};
        try{
            S.O.P(a[200]); //throw new ArrayIndexOutOfBoundsException
        }
        catch (ArrayIndexOutOfBoundsException e){
            S.O.P("Invalid index");
        }
    }
}
//Exception e=new ArrayIndexOutOfBoundsException();
//ArrayIndexOutOfBoundsException e = new ArrayIndexOutOfBoundsException();
```

Internal working when we divide a number by zero

* When we divide a number by zero internally an object of ArithmeticException is thrown.
* The object thrown could be caught using same type (ArithmeticException) or superclass type (Exception).

```
package com;
class test{
    psvm(){
```

```
//Specialization -> ArithmeticException e = new ArithmeticException();
    try{
        S.O.P(10/0); //throw new ArithmeticException();
    }
    Catch(ArithmeticException e){
        S.O.P("Invalid Denominator");
    }
    //Generalization -> Exception e = new ArithmeticException();
    try{
        S.O.P(10/0); //throw new ArithmeticException.
    }
    catch(Exception e){
        S.O.P("Invalid Denominator");
    }
}
}
```

Note: * One try block can have multiple catch blocks & that suitable catch block will get executed.

* It is always a good practice to handle the super class catch block as the last catch block.

* We cannot have any executable lines of code b/w try block & catch block

---

Ex:-
```
public class Demo
{
    psvm()
    {
        try{
            S.O.P(10/0);
        }
        catch (NullPointerException e){
            S.O.P("Some Null Exception");
        }
        catch(ArithmeticException e){
            S.O.P("Invalid Denominator");
        }
        catch(Exception e){
            S.O.P("Super Class Handler");
        }
    }
}
```

O/P:-

Invalid Denominator

Types of Exception:-

1. Check Exception
2. Uncheck Exception

Check Exception:-

* Check Exceptions are compiler known exceptions & compiler will force you to handle immediately.

* All the Check Exception will inherit Exception class

Ex:- ClassNotFound, InterruptedException, FileNotFoundException

Unchecked Exception:-

* Unchecked Exception are compiler unknown exceptions & compiler will not force you to handle immediately.
* All the Unchecked Exception will inherit RuntimeException.

Ex:- ArithmeticException, ArrayIndexOutOfBoundsException, etc

Example for Check Exception

```
import java.io.*;
Class A{
  psvm(){
   for(int i=1;i<=5;i++){
      S.O.P(i);
      try{
         Thread.sleep(2000); //pause exception for 2 secs
      }
      catch(InterruptedException e){
         S.O.P("Handled");
      }
   }
   try{
      FileReader f= new FileReader("guldu.txt");
   }
   catch(FileNotFoundException e){
      S.O.P("File Not Found");
   }
}
}
```

O/P:-
1
2
3
4
5
File Not Found

Exception → Check → Compiler known → Handle immediately → extends Exception

Exception → Uncheck → Compiler unknown → Handle, not immediately → extends RuntimeException

Important methods present in Throwable Class

1) getMessage():-

- getMessage() is used to return the message about the exception.

Syntax:-

```
public String getMessage()
```

Ex:-

```
package com;
class MethodExample
{
    public static void main(String[] args)
    {
        try
        {
            int[] a = {10,20};
            S.O.P(a[99]);
        }
        catch(Exception e)
        {
            S.O.P(e.getMessage());
            String message = e.getMessage();
            S.O.P(message);
        }
    }
}
```

O/P:-

Index 99 out of bound length 2
Index 99 out of bound length 2

2) printStackTrace():-

printStackTrace() will print the complete information about the exception.

Syntax:-

```
public void printStackTrace()
```

Ex:-

```
class MethodExample
{
    p s v m ()
    {
        S.O.P("start");
        try
        {
            S.O.P(10/0);
        }
        catch (Exception e)
        {
            e.printStackTrace(); //prints ExceptionName, line No & Message
        }
        S.O.P("end");
    }
}
```

O/p:- start
java.lang.ArithmeticException: / by zero
at com.MethodExample.main(MethodExample.java
end

throws:-

* throws is a keyword which is used to indicate the caller about the possibility of an exception.
* throws can be used with both check exception and uncheck exception but majorly used with check exception.
* throws can be used with both methods and constructors but majorly used with method declaration.
* We can declare multiple exceptions with the help of throws.

Ex:-1)

```
class ThrowsDemo1{
    static void divide() throws ArithmeticException
    {
        S.O.P(10/0);
    }
    p s v m ()
    {
```

```
        S.O.P("start");
        try
        {
            divide();
        }
        catch(ArithmeticException e)
        {
            S.O.P("Invalid Denominator");
        }
        S.O.P("end");
    }
}
```

O/P:-
start
Invalid Denominator
end

Ex-2)
```
class ThrowsDemo2
{
    static void display()throws Exception
    {
        for(int i=1; i<=5; i++)
        {
            S.O.P(i);
            Thread.sleep(1000);
        }
    }
    p s v m:()
    {
        try
        {
            display();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

O/P:-
1
2
3
4
5

Ex-3)
```
class ThrowsDemo3
{
    static void readDataFromFile()throws FileNotFoundException
    {
        FileReader f=new FileReader("dinga.txt");
```

```
//S.O.P(10/0);
}
psvm()
{
    try
    {
        readDataFromFile();
    }
    catch(FileNotFoundException e)
    {
        S.O.P("File Not Present");
    }
}
}
```

O/P:-
File Not Present

Note:-

* throws will increase the efficiency of a programs by avoiding the redundancy.
* We should handle an exception only when it is necessary, with the help of throws we can achieve this concept.
* It is possible to use throws with main method but it is not a good practice as the caller of main method is JVM and it is not possible to handle an exception at the JVM side.
* With the help of throws, we can declare multiple exceptions.

Ex:-
```
void throws ArithmeticException, NullPointerException {
}
void throws Exception {
}
```

Custom Exception / User Defined Exception

* Based on the project ~~if there is~~ it is sometimes necessary to create an user defined exception.
* Any exception which the user/programmer creates is called as custom exception.

Rules for developing custom exception

* Create a class with the exception name.
* Class should inherit RuntimeException class to create unchecked exception or class should inherit Exception class to create check exception.
* Optionally Override getMessage().
* Invoke the exception object using throw keyword, handle it using try and catch block and provide suitable solution.

throw keyword :-

* throw is a keyword which is used to invoke an object of exception
* throw is generally used with custom exception

Syntax :-

```
throw objectOfExceptionType;
        or
throw new ClassName();
```

Ex: (for Custom Exception)

```
package customexception;
class InvalidPasswordException extends RuntimeException {

}
```

```
package customexception
import java.util.Scanner;
class login
{
    p s v m ()
    {
        Scanner sc= new Scanner(System.in);
        S.O.P("Enter Password:");
        int password = sc.nextInt();
        if(password ==123)
        {
            S.O.P("Login Successful");
        }
        else
        {
            try
            {
                throw new InvalidPasswordException();
            }
            catch(InvalidPasswordException e)
            {
                S.O.P("Please Enter valid Password!");
            }
        }
    }
}
//InvalidPasswordException e= new InvalidPasswordException();
//Exception e = new InvalidPasswordException();
```

Q) Develop an uncheck exception called as InsufficientBalanceException and override getMessage method.

```
package customexception
class InsufficientBalanceException extends RuntimeException{
    private String message;
    InsufficientBalanceException(String message)
    {
        this.message = message;
    }
    @Override
    public String getMessage()
    {
        return message;
    }
}
import java.util.Scanner;
class ATM
{
    p s v m ()
    {
        Scanner sc= new Scanner(System.in);
        int balance = 5000;
        S.O.P ("Enter amount to be withdrawn:");
        int amount = sc.nextInt();
        if(amount < balance)
        {
            S.O.P("Amount Withdrawn Successfully");
        }
        else
        {
            try{ InsufficientBalanceException obj=new InsufficientBalanceException("No Funds");
                throw obj;
                or
                throw new InsufficientBalanceException("Insufficient Funds!");
            }
            catch(Exception e){
                S.O.P(e.getMessage());
            }
        }
    }
}
```

Q) Develop a check Exception called as AgeInvalidException and Override getMessage method.

```
package customexception
class AgeInvalidException extends Exception
{
    private String message;
    AgeInvalidException(String message)
    {
        this.message=message;
    }
    public String getMessage()
    {
        return message;
    }
}
import java.util.Scanner;
class Vote
{
    p s v m()
    {
        Scanner sc=new Scanner(System.in);
        S.O.P("Enter Person Age:");
        int age=sc.nextInt();
        if(age>=18)
        {
            S.O.P("Eligible to vote");
        }
        else
        {
            try
            {
                throw new AgeInvalidException("Not Eligible");
            }
            catch(Exception e)
```

O/P:
Enter Person Age:
16
Not Eligible

```
        {
            S.O.P(e.getMessage());
        }
    }
}
}
```

finally block:-

* finally block is a block of code / set of instructions which gets executed always, even if an exception occurs or not.

Syntax:-

```
finally {
    [ ≡ ]
}
```

* The important lines of code which should get executed always, should be written inside finally block.

Ex:-

```
class Demo
{
    psvm()
    {
        S.O.P("start");
        try{
            S.O.P(10/0);
        }catch(ArithmeticException e){
            S.O.P("Invalid Denominator");
        }
        finally{
            S.O.P("Inside Finally Block");
        }
        S.O.P("end");
    }
}
```

O/P:-

```
Start
Invalid Denominator
Inside Finally Block
end
```

Note:-

① It is possible to have nested try and catch block in java.

Syntax:
```
try{
    try{
    }
    catch(Exception e){
    }
}
catch(Exception e){
}
```

② It is possible to have try & catch block within finally block.

Syntax:
```
finally{
    try{
    }
    catch(Exception e){
    }
}
```

③ It is possible to have try without catch block when we have try and finally block

Syntax:
```
try{
}
finally{

}
```

④ First exception needs to be avoided & then handle it.

Ex:
```
//Avoiding NullPointerException
String s="java";
if(s!=null){
    S.O.P(s.length());
}
```

~~Collection Framework~~ Collection Framework

Data Structure:-

Data structure is a way of storing / arranging / organizing the data.

ex:- Stack, Queue, Array, Linked List, Graphs, Trees

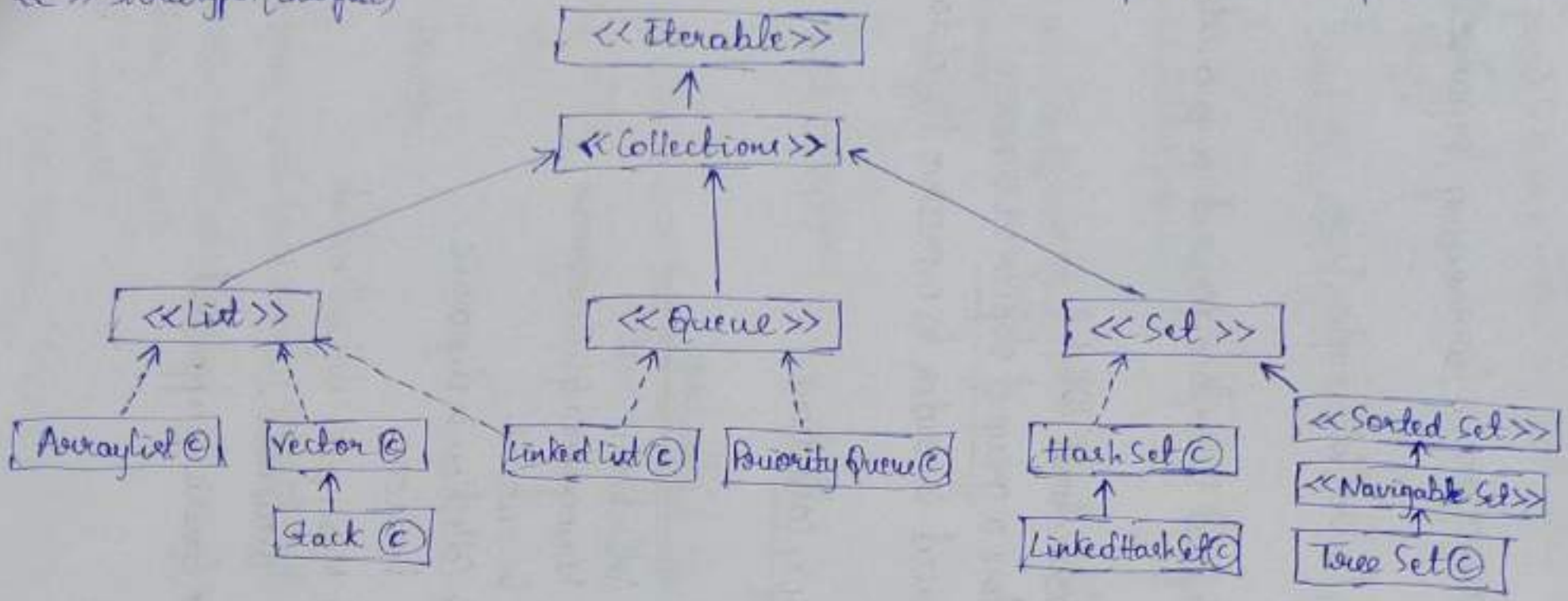
Collection:-

* Collection is a pre-defined interface present in java.util package.
* Collection was introduced from JDK 1.2
* Collection is used to store a group of objects or elements.
* Collection was introduced in order to overcome the disadvantages of arrays.

Difference between array & collections

Array	Collection
* Array is homogeneous in nature.	* Collection is both homogeneous & heterogeneous in nature.
* Array is fixed size.	* Collection is dynamic in size.
* No pre-defined methods available.	* Many pre-defined methods available.
* Generics not supported.	* Generics Supported.

Collection Hierarchy



Important methods present in Collection interface

① **add():-**
add() is used to add an object into the collection.

② **addAll():-**
addAll() is used to add one collection into another collection.

③ **remove():-**
remove() is used to remove an object from the collection based on index position.

④ **removeAll():-**
removeAll() is used to remove one collection from another collection.

⑤ **contains():-**
contains() is used to check if an object is present in the collection or not.

⑥ **containsAll():-**
containsAll() is used to check if one collection is present in another collection or not.

⑦ **size():-**
size() is used to calculate number of objects present in collection

⑧ **isEmpty():-**
isEmpty() is used to check if the collection is empty or not.

⑨ **clear():-**
clear() is used to remove all the objects from the collection.

⑩ **public Iterator iterator():-**
iterator() is used to traverse/iterate a group of objects from the collection.

List:-

* List is a pre-defined interface present in java.util package.
* List was introduced from JDK 1.2.
* The implementation classes of List interface are as follows
  - → ArrayList
  - → Vector
  - → Stack
  - → LinkedList

```
List l = new ArrayList()
         new Vector()
         new Stack()
         new LinkedList()
```

Generalization / Upcasting

- **Specifications of List interface**
  - → Insertion Order is maintained.
  - → Duplicates are allowed.
  - → List is index based
  - → Null values can be inserted.

Important methods present in list interface

1) **get():-**

   get() is used to return the object based on index position.

2) **indexOf():-**

   indexOf() is used to find the index position of an object.

3) **lastIndexOf():-**

   lastIndexOf() is used find the last index position of an object.

4) add(index, value)

5) set(index, value)

File Handling / File Programming

The data or information from a java program can be stored onto platforms persistently

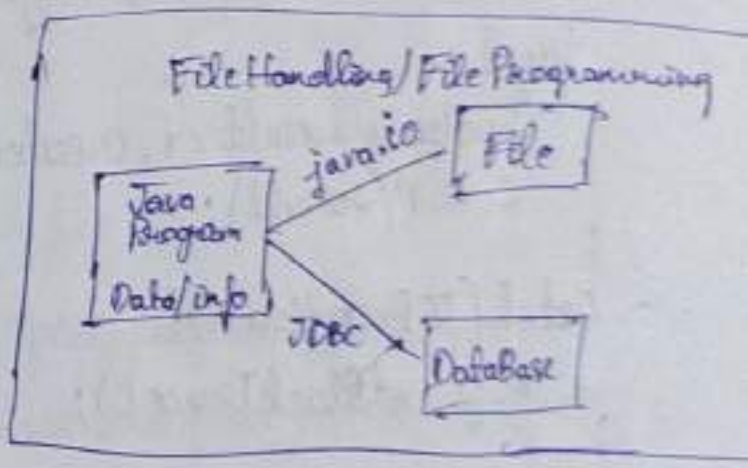
JDBC - Java Database Connectivity

1) In Files using java.io package
2) Database using JDBC

The process of reading the data from a file & writing the data into a file is called as file handling and file programming

Import classes present in java.io package

1) File
2) FileReader
3) FileWriter
4) FileOutputStream
5) FileInputStream
6) ObjectOutputStream
7) ObjectInputStream



Program to create a folder/directory

```
package com;
import java.io.File;
class Demo{
    p s v m ()
        //Object Representation of a file or folder
        File f = new File("c:/gulabu");
        boolean result = f.mkdir();
```

O/P:-
true

```
        S.O.P(result);
    }
}
```

Note:-

1) mkdir() is used to create a folder in the specified path

Program to create a file

```
package com;
import java.io.File;
import java.io.IOException;
class Test {
    P S V m() {
        File f = new File("e:/dinga.txt");
        try {
            boolean result = f.createNewFile();
            S.O.P(result);
        }
        Catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

O/P:-
true

Note:-

1) CreateNewFile() is going to create a new File based on extension specified (.txt, .pdf, .doc, .exe, .xlsx)

exists():-

This method is used to check if the specified file/folder is present or not.

delete():-
delete() is used to delete a specified file/folder

Ex:-
```
package com;
import java.io.File;
class Runner{
    psvm(){
      //Searches for guldu folder within project, as path is not specified
      File f= new File("guldu");
      S.O.P(f.exists());
      f.delete();
      S.O.P(f.exists());
    }
}
```

O/P:- true
false

Steps to store/write the data into a file

1) Create an instance of File class by specifying the file name.
2) Create an instance of FileWriter class and pass the object of File to the constructor of FileWriter.
3) Invoke the write method, which is converting the data into streams.
4) Invoke the flush method which stores the data into the file.
5) Close the FileWriter Connection within finally block.

Ex:-
```
package com;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

```
class WriteDemo{
    p s v m() {
        File f = new File("demo.txt");
        FileWriter fw = null;
        try {
            fw = new FileWriter(f);
            fw.write("Dabba Fellows");
            fw.flush();
            S.O.P("Data Stored");
        }
        catch(IOException e){
            e.printStackTrace();
        }
        finally{
            try{
                fw.close();
            }
            catch(IOException e){
            e.printStackTrace();
            }
        }
    }
}
//FileWriter fw = new FileWriter(new File("demo.txt"));
```

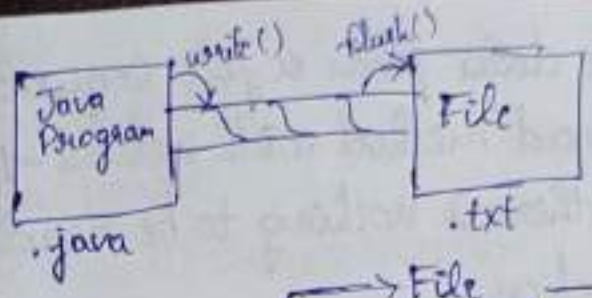
O/P:
Data Stored

In demo.txt
O/P: Dabba Fellows

Note:-

FileWriter():-

* If the file is existing, FileWriter() will open the existing file in overridden manner (open the file in 'write' mode).
* If the file is not present, FileWriter() will dynamically create a new File and open the file in 'write' mode.



A → 65 → 01101111
Stream

FileWriter
- → File present → open the existing file in overridden manner → & opens in 'write' mode
- → File Not present → Creates a new File → open in 'write' mode

Ex:-
```
import java.io.*;
class WriteExample
{
    psvm() throws Exception
    {
        File f = new File("dinga.txt");
        FileWriter fw = new FileWriter(f);
        fw.write("Chicken Biryani");
        fw.flush();
        fw.close();
        S.O.P("Data written successfully");
    }
}
```

O/P:-
Data written successfully

In dinga.txt:-
O/P:- Chicken Biryani

//FileWriter fw = new FileWriter(f, true); → it appends/concatenate the content of file.

Steps for Reading the data from a file

1) Create an object/instance of file by passing the file name.
2) Create an object of FileReader and pass the file object.

FileReader (opens the file in read mode)
- → File present → open the file in 'read' mode
- → File Not present → FileNotFoundException

3) Define a business logic to read the data from a file sequentially with the help of read method (read method will return -1 when the file is empty or when there is nothing to be read).

4) Optionally close FileReader connection.

Program:-

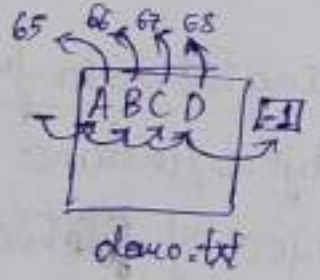
```
import java.io.*;
class ReadDemo
{
    PSVM()
    {
        File f = new File("demo.txt");
        try {
            FileReader fr = new FileReader(f);
            int ch = fr.read();
            while (ch != -1) {
                S.O.P((char)ch);
                ch = fr.read();
            }
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

O/P:-
ABCD

Tracing:-

```
int ch = fr.read();
while(ch != -1)
{
    S.O.P((char)ch)
    ch = fr.read();
}
```

ch: 65, 66, 67, 68, -1



A
B
C
D

Programs:

```
import java.io.*;
class ReadDemo1
{
    P S V M ()-throws Exception
    {
        File f= new File("example.txt");
        FileReader fr= new FileReader(f);
        int x= fr.read();
        while(x!=-1)
        {
            S.O.P((char)x);
            x= fr.read();
        }
        fr.close();
    }
}
```

O/P:-

File Handling

Session @ 10:3
at
Basavanagudi

Serialization & Deserialization

* In java objects will get created inside heap area, where in it is not permanently stored therefore in order to store objects into a file we use serialization & deserialization concept.
* The process of storing or writing the objects inside a file is called as Serialization.
* The process of reading the objects from a file is called as deserialization.

Note:-

→ Class should implement serializable interface
→ Objects should be stored inside a file with the extension .ser

Program:-

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
— // — ObjectInputStream;
— // — ObjectOutputStream
— // — Serializable;

class Student implements Serializable {
    int age;
    Student(int age){
        this.age=age;
    }
    @Override
    public String toString(){
        return "Age:"+age;
    }
}
class Solution {
    psvm() throws Exception {
        FileOutputStream fos= new FileOutputStream("Student.ser");
        ObjectOutputStream oos= new ObjectOutputStream(fos);
        Student s= new Student(22);
        oos.writeObject(s); //oos.writeObject(new Student(22));
        oos.flush();
        fos.close();
        oos.close();
        S.O.P("Object Serialized");
        FileInputStream fis = new FileInputStream("Student.ser");
        ObjectInputStream ois =new ObjectInputStream(fis);
```

```
//Object obj = new Student(22);
Object obj = ois.readObject(); //new Student(22);
Student std = (Student)obj; //Student std = (Student) ois.readObject();
fos.close();
ois.close();
s.o.p(std);
s.o.p("Object Deserialized");
}
}
```

o/p:-
Object Serialized
Age: 22
Object Deserialized

6) Write a java program to serialize & deserialize employee objects with respect to file where in employee will have id & name.

Sol

```
import java.io.*;
class Employee implements Serializable
{
    int id;
    String name;
    Employee(int id, String name){
        this.id = id;
        this.name = name;
    }
    @Override
    public String toString(){
        return "ID:"+id+"\t Name:"+name;
    }
}
class Solution
{
    p s v m ( ) throws Exception
    {
```

```
FileOutputStream fos = new FileOutputStream("Employee.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(new Employee(101, "Tom"));
oos.flush();
fos.close();
oos.close();
S.O.P("Object Serialized");
FileInputStream fis = new FileInputStream("Employee.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
Object obj = ois.readObject(); //new Employee(101, "Tom");
Employee emp = (Employee) obj; //Employee emp = (Employee) ois.readObj
fis.close();
ois.close();
S.O.P(emp);
S.O.P("Object Deserialized");
}
}
```

O/P:-

```
Object Serialized
ID: 101 Name: Tom
Object Deserialized
```

Note:-

transient variables or data members cannot be serialized

ex:- transient int id;

Collections

ArrayList :-

* Arraylist is a predefined class present in java.util package.
* Arraylist was introduced from JDK 1.2.
* The initial capacity of Arraylist is 10.
* The incremental capacity of Arraylist is $\left(\frac{\text{current capacity} \times 3}{2}\right) + 1$
* The underlined data structure of Arraylist is resizable array or growable array.

Note:- All the 4 specifications of List interface is followed by Arraylist as well.

Working of Arraylist:-

→ When we create an instance/object of Arraylist, internally an array will get created based on the initial capacity or custom capacity.

→ When we add an object into Arraylist, once it is full, a new array gets created based on the incremental capacity formula

→ All the objects of old array gets copied into the newly created array.

→ The reference variable which was pointing to the old array gets dereferenced & starts pointing to the newly created array.

Ex:-
```
package days;
import java.util.Arraylist;
class Demo{
    p s v m ( ){
        Arraylist l= new Arraylist ( );
        l.add(10);
```

```
l.add(20.5);
l.add("java");
l.add(10);
l.add(null);
S.O.P(l); //[10,20.5,java,10,null]
S.O.P(l.get(1)); //20.5
S.O.P(l.size()); //5
S.O.P(l.contains("Java")); //false
S.O.P("------");
S.O.P(l); //[10,20.5,java,10,null]
l.remove(1);
S.O.P(l); //[10,java,10,null]
S.O.P("------");
S.O.P(l.isEmpty()); //false
l.clear();
S.O.P(l.isEmpty()); //true
}
}
```

9) Write a java program to store 3 heterogeneous elements into Arraylist and traverse it both in forward & reverse direction using for loop.

Sol

```
class TraversingArraylist
{
    public static void main(String[] args)
    {
        Arraylist l=new Arraylist();
        l.add(10);
        l.add("java");
        l.add(20.5);
```

```
        for(int i=0; i< l.size(); i++)
        {
            S.O.P (l.get(i));
        }
        S.O.P("----"); l.size()-1
        for(int i= ; i>=0 ; i--)
        {
            S.O.P (l.get(i));
        }
    }
}
```

length → Array
length() → String
size() → Collection

O/P:-
10
java
20.5

20.5
java
10

Note:- ***

Arraylist implements the following interfaces
-> List
-> RandomAccess
-> Cloneable
-> Serializable

Program:-

```
import java.util.ArrayList;
class Runner
{
    public static void main (String[] args)
    {
        Arraylist l = new ArrayList();
        l.add(15);
        l.add(28);
        l.add(32);
        S.O.P (l); //[15,28,32]
    //adding 55 in index position 1, & already existing object gets shifted
        l.add(1,55);
        S.O.P (l); //[15,55,28,32]
```

```
//adding 77 in index position 2, & already existing object gets overrid
    l.set(2,77);
    S.O.P(l);  //[15,55,77,32]
  }
}
```

Constructors present in Arraylist

→ Arraylist()
→ Arraylist (int initialCapacity)
→ Arraylist (Collection c)

Program:-

```
import java.util.Arraylist;
class MainClass
{
  p s v m ()
    //Internally an array gets created of default initial capacity(10)
      Arraylist l1=new Arraylist();
    //Internally an array gets created of custom initial capacity(20)
      Arraylist l2=new Arraylist(20);
      Arraylist al=new Arraylist();
      al.add(10);
      al.add(20);
    //adding 1 collection into another collection
      Arraylist lst = new Arraylist(al);
      lst.add(30);
      S.O.P(al); //[10,20]
      S.O.P(lst); //[10,20,30]
  }
}
```

O/P:-
[10,20]
[10,20,30]

Linked List:-

* Linked List is a predefined class present in java.util package.
* It was introduced from JDK 1.2. * Objects are stored in the form of nodes
* LinkedList doesn't have any initial capacity / incremental capacity
* The objects inside linked list is stored in the form of nodes where in each node will store the object, address of previous node & address of next node.
* Underlined data structure of linked list is doubly linked list.

Linked list implements the following interfaces:-

→ List
→ Dequeue
→ Cloneable
→ Serializable

Constructors in LinkedList:-

→ LinkedList()
→ LinkedList(Collection c)

Program:-

```
import java.util.LinkedList;
class Demo{
    psvm(){
        LinkedList l=new LinkedList();
        l.add(10);
        l.add(20.5);
        l.add(10);
        l.add("java");
        l.add(10);
        l.add(true);
        S.O.P(l);
```

```
        S.O.P(l.size());
        S.O.P(l.indexOf(10));
        S.O.P(l.lastIndexOf(10));
    }
}
```

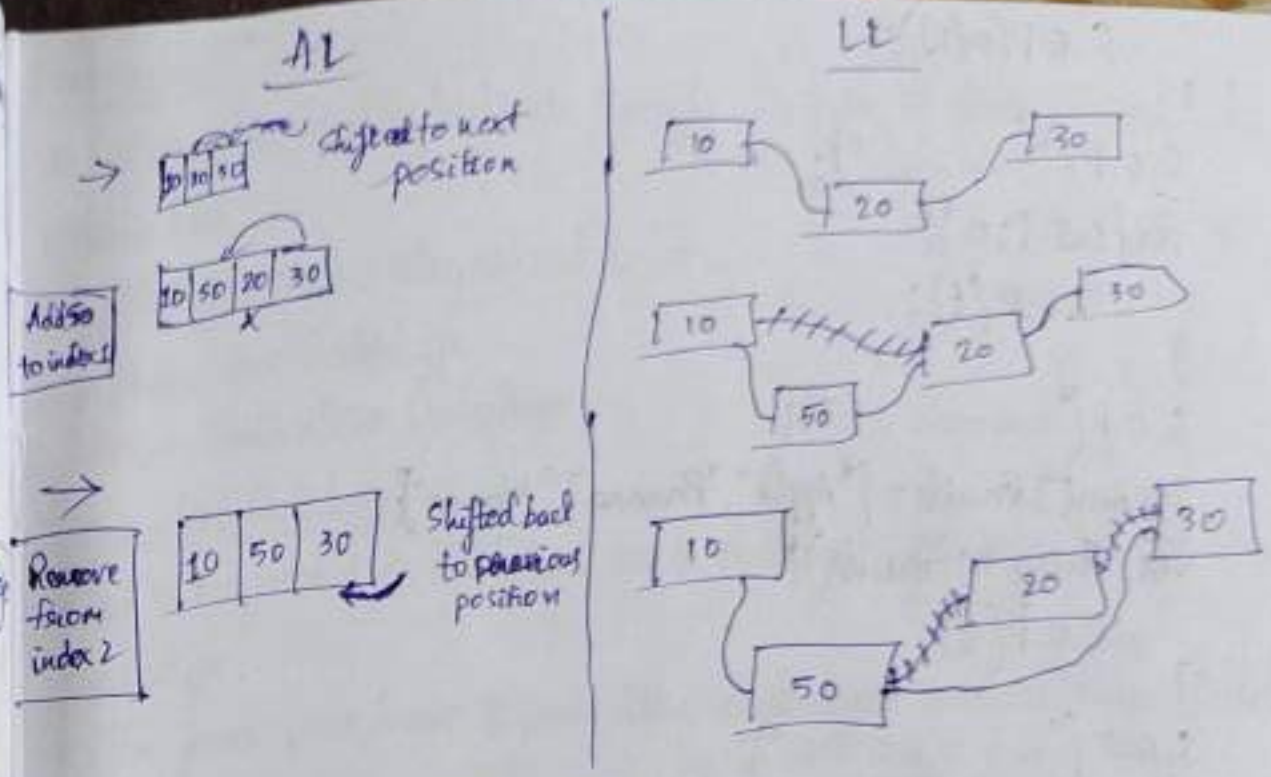
O/P:-
[10, 20.5, 10, java, 20, ...]
6
0
4

Difference between Arraylist & Linked List

Arraylist	Linked List
* Predefined class in java.util package introduced from JDK 1.2	* Predefined class in java.util package introduced from JDK 1.2
* Initial Capacity is 10.	* No initial capacity concept.
* Incremental Capacity is $\left(\frac{\text{Current Capacity} \times 3}{2}\right)+1$	* No incremental capacity concept.
* Underlined data structure is resizeable/growable array.	* Underlined data structure is doubly linked list.
* 3 constructors.	* 2 constructors.
* Sequential memory allocation	* Non-sequential memory allocation
* Arraylist implements List, RandomAccess, Cloneable, Serializable.	* Linked list implements List, Deque, Cloneable, Serializable.

Note:-

* We prefer LinkedList over Arraylist when there is lot of insertion & deletion in between because in Arraylist shifting of objects is involved which reduces efficiency where as in LinkedList shifting is not involved where in efficiency is higher.
* We prefer Arraylist over LinkedList when there is storing & retrieving of objects involved because Arraylist is sequential and traversing is faster where as LinkedList is not sequential where in traversing is slower.



Enhanced For loop or For Each loop :-

* Enhanced For loop is generally used to traverse a group of objects from a collection / array.

Syntax:-

```
~~for (datatype variable~~
for (DatatypeToBeTraversed variable : Array/Collection)
{

}
```

Note :- * Enhanced For Loop internally make use of iterator.
* Enhanced For Loop is uni-directional.

Program:-

```
package day3;
class EnhancedForLoopDemo {
        psvm ()
            int[] a = {10,20,30};
            for (int i = 0; i < a.length; i++){
```

```
        S.O.P(a[i]);
    }
    S.O.P("------");
    for(int i:a){
        S.O.P(i);
    }
    S.O.P("========");
    String[] fruits={"Apple","Banana","Mango"};
    for(String x:fruits){
        S.O.P(x);
    }
    S.O.P("========");
    double[] salaries={1.2,3.4,5.6,7.8};
    for(double sal:salaries){
        S.O.P(sal);
    }
  }
}
```

O/P:-

```
10
20
30
------
10
20
30
========
Apple
Banana
Mango
========
1.2
3.4
5.6
7.8
```

Datatypes:-

* Datatype is used to indicate / specify the type of data stored into a variable.
* Datatypes are categorized into.
  → Primitive Datatype
  → Non-Primitive Datatype

Primitive Datatype:-

* If a keyword is behaving as a datatype, we refer it as primitive datatype.
* In java, we have 8 primitive datatypes and all they have fixed size.

Datatype	Memory Size	Default value
byte	1 byte - 8 bits	0
short	2 byte - 16 bits	0
int	4 byte - 32 bits	0
long	8 byte - 64 bits	0
float	4 byte - 32 bits	0.0
double	8 byte - 64 bits	0.0
boolean	True/false (1 bit)	false
char	2 bytes - 16 bit)	unicode / non printable character

Non Primitive Datatype:-

* When a class / interface behaves as a datatype, we refer it as non-primitive datatype.
* In java we can have 'n' number of non-primitive datatypes & they do not have fixed size.
* Default value for any non-primitive datatype is null.

Collection Framework doesnot support primitive datatype, the alternative used is called as Wrapper class.

Wrapper class:-

* The Non primitive version of primitive datatype is called as Wrapper class.
* All the wrapper class & string class are present in java.lang package.
* toString() is overridden in all wrapper classes and string class.

Primitive Datatype	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Auto Boxing & Auto Unboxing:-

* The process of converting primitive type to non-primitive type is called as Auto Boxing.
* The process of converting non-primitive type to primitive type is called as Auto Unboxing.

Ex: for Auto Boxing

Ex1:-
```
double c = 10.5;
Double d = new Double(c);
S.O.P(c+" "+d) //10.5 10.5
```

Ex2:-
```
char i = 'A';
Character j = new Character(i);
S.O.P(i+" "+j); //A A
```

Example for Auto Unboxing:-

Ex1:-
```
Integer a = new Integer(25);
int b = a;
S.O.P (a+" "+b); //25 25
```

Ex2:-
```
Character x = new Character('Z');
char y = x;
S.O.P(x+" "+y); //Z Z
```

Generics:-

* Generics is used to specify the Element Type of a Collection
* Generics was introduced from JDK 1.5

Syntax:-

<Element Type>

« »	< >
Stereotypes for (interfaces)	Generics for (Element Type)

Program:-
```
package day4;
import java.util.ArrayList;
import java.util.LinkedList;
```

```
class GenericsDemo
{
    p s v m ( ){
        Arraylist <String> al = new Arraylist <String>();
        al.add("java");
        al.add("movie");
        al.add("hello");
        for(String s : al){
            S.O.P(s);
        }
        S.O.P("-------");

        Arraylist <Integer> l = new Arraylist <Integer>();
        l.add(10);
        l.add(15);
        l.add(20);
        for(Integer i : l) {//int
            S.O.P(i);
        }
        S.O.P("-----");
        Linkedlist ll = new linked list();
        ll.add(10);
        ll.add("java");
        ll.add(true);
        for(Object obj : ll){ //Generalization -> upcasting
            S.O.P(obj);
        }

    }
}
```

O/P:-

```
java
movie
hello
-------
10
15
20
-------
10
java
true
```

Program:

```
import java.util.ArrayList;
class Test{
    psvm(){
        int a=10;
        Double b=1.5;
        ArrayList l=new ArrayList();
        l.add(a); //l.add(new Integer(10));
        l.add(b); //l.add(new Double(1.5));
        l.add('A'); //l.add(new Character('A'));

        //Generalization -> Upcasting
        //Object obj=new Integer(10),new Double(1.5),new Character('A');
        for(Object obj : l){
            S.O.P(obj);
        }
    }
}
```

O/P:-

```
10
1.5
A
```

Collection methods implementation program

```
import java.util.*;
class Runner{
    psvm(){
        ArrayList al=new ArrayList();
        al.add(10);
        al.add(20);
        LinkedList ll=new LinkedList();
        ll.addAll(al);
        ll.add(30);
        S.O.P(ll.containsAll(al));
        S.O.P(al); //[10,20]
        S.O.P(ll); //[10,20,30]
        ll.removeAll(al);
        S.O.P(ll); //[30]
    }
```

O/P:-

```
true
[10,20]
[10,20,30]
[30]
```

Write a java program by following below scenarios

a) Create a class called as Student declare 2 attributes called as Age & Name.

b) Initialize these 2 attributes with the help of 2 constructors

c) Override toString method to return age & name in the below format (Age of Tom is 20).

Sol

```
import java.util.Arraylist;
class Student
{
    int age;
    String name;
    Student(int age, String name)
    {
        this.age = age;
        this.name = name;
    }
    @Override
    public String toString()
    {
        return "Age of " + Name + "is" + age;
    }
}
class Solution
{
    p s v m ()
    {
        Student s1 = new Student (21, "Tom");
        Student s2 = new Student (22, "Jack");
        Student s3 = new Student (23, "Smith");
        Arraylist <Student> l = new Arraylist <Student> ();
        l.add(s1);
        l.add(s2);
```

```
        l.add(s3);
        for(int i=0;i<l.size();i++){
            S.O.P(l.get(i)); //l.get(i).age if toString() is not overridden
        }
        S.O.P("------");
        for(Student s:l){
            S.O.P(s); //S.O.P(s.age+" "+s.name);
        }
    }
}
```

O/P:
```
Age of Tom is 21
Age of Jack is 22
Age of Smith is 23
------
Age of Tom is 21
Age of Jack is 22
Age of Smith is 23
```

Q) Write a java program to store 3 employee objects inside LinkedList and traverse it using enhanced for loop after overriding toString().

Sol
```
package day4;
class Employee
{
    int id;
    double salary;
    String name;
    Employee(int id, double salary, String name){
        this.id=id;
        this.salary=salary;
        this.name=name;
    }
    @Override
    public String toString()
    {
```

```
    return "Id of"+name+" and salary is"+salary;
  } return
}
class Solution
{
    psvm(){
        Employee e1 = new Employee(101, 25000, "Tom");
        Employee e2 = new Employee(102, 30000, "Jack");
        Employee e3 = new Employee(103, 35000, "Smith");
        LinkedList<Employee> ll = new LinkedList<Employee>();
        ll.add(e1);
        ll.add(e2);
        ll.add(e3);
        for(Employee e : ll){
            S.O.P(e);
        }
    }
}

    psvm()
    {
        Linkedlist<Employee> l = new linkedlist<Employee>();
        Employee e = new Employee(10, 1.5, "A");
        l.add(e);
        l.add(new Employee(20, 2.5, "B"));
        for(Employee emp : l){
            S.O.P(emp);
        }
    }
}
```

```
class Mobile
{
    String brand;
    int cost;
    Mobile(String brand, int cost)
    {   this.brand=brand;
        this.cost=cost;
    }@Override
    public String toString()
    {
        return "Cost of"+brand+"is"+cost;
    }
}import java.util.ArrayList;
class MainClass
{
    public static void main(String[] args)
    {
        ArrayList<Mobile> l = new ArrayList<Mobile>();
        Mobile m=new Mobile("Apple", 55000);
        l.add(m)
        l.add(new Mobile("Samsung", 45000));
        l.add(new Mobile("Vivo", 15000));
        l.add(new Mobile("Oppo", 15000));
        l.add(new Mobile("Lava", 12000));
        for(Mobile mob: l)
        {
            S.O.P(mob);
        }
        S.O.P("------");
        for(Mobile mob: l){
            if(m.cost >= 10000 && m.cost<=20000){
                S.O.P(mob);
            }
        }
    }
```

O/P:-

```
Cost of Apple is 55000
Cost of Samsung is 45000
Cost of Vivo is 15000
Cost of Oppo is 15000
Cost of Lava is 12000
------
Cost of Vivo is 15000
Cost of Oppo is 15000
Cost of Lava is 12000
```

Vector:-

* Vector is a pre-defined class present in java.util package
* Vector was introduced from JDK 1.0, therefore vector is referred as legacy collection.
* Initial capacity of vector is 10.
* Incremental capacity of vector is "current capacity * 2".
* Underlined data structure of vector is Resizable Array/ Growwable Array.
* Vector is Thread Safe.
* Constructors present in vector are as follows:

→ Vector()
→ Vector (int initialCapacity)
→ Vector (int initialCapacity, int incrementalCapacity)
→ Vector (Collection c)

Difference b/w Vector & ArrayList

Vector	ArrayList
* Introduced in JDK 1.0	* Introduced in JDK 1.2
* Incremental Capacity is Current Capacity * 2	* Incremental Capacity is $\left(\frac{\text{Current Capacity} * 3}{2}\right) + 1$
* Vector is Thread Safe	* Arraylist is not Thread Safe
* It has 4 constructors	* It has 3 constructors

Note:-
Vector implements the following interfaces:-

→ List
→ RandomAccess
→ Cloneable
→ Serializable

Internal Working of Vector:-

* When we create a object of vector internally an array gets created of size with initial capacity or custom capacity.
* When we add objects to vector & once it is full, a new array gets created.
* Then the objects in old array gets stored onto the newly created array and the reference variable ~~object~~ pointing to it gets dereferenced.
* Then it starts pointing to the newly created array.

Q) WAJP to store 3 heterogeneous elements into vector and traverse it using enhanced for loop.

Sol

```
class Vector
{
    p s v m ()
    {
        Vector v = new Vector();
        v.add(10);
        v.add(12.5);
        v.add("java");
        v.add(true);
        for(Vector vec : v){
            S.O.P(vec);
        }
    }
}
```

O/P:-
10
12.5
java
true

Difference b/w Collection & Collections

* Collection is a pre-defined interface present in java.util package introduced from JDK 1.2.
* Collections is a pre-defined class present in java.util package introduced from JDK 1.2.

Sort():-

Sort() is a static method present in Collections class which is used to sort a list of homogeneous objects.

Syntax:- sort(List l) & sort(List l, Comparator c)

Ex:-

```
import java.util.ArrayList;
import java.util.Collections;
class Demo{
    psvm(){
        ArrayList<Integer> l = new ArrayList<Integer>();
        l.add(20);
        l.add(30);
        l.add(10);
        S.O.P("Before Sorting:");
        for(int i: l){
            S.O.P(i); //20,30,10
        }
        Collections.sort(l);
        S.O.P("After Sorting: Ascending Order");
        for(int i: l){
            S.O.P(i); //10,20,30
        }
        Collections.reverse(l);
        S.O.P("Descending Order");
        for(int i: l){
            S.O.P(i); //30,20,10
        }
    }
}
```

O/P:-

```
Before Sorting
20
30
10
After Sorting: Ascending Order
10
20
30
Descending Order
30
20
10
```

Stack:-

* Stack is a predefined class present in java.util package.
* Stack was introduced from JDK 1.0, therefore we refer stack as as legacy collection.
* Stack follows LIFO traversing order (Last In First Out).
* The process of adding an object into the stack is called as 'push' & removing an object from the stack is called as 'pop'.

Program:-

```
import java.util.Stack;
class Demo{
    psvm(){
        Stack s=new Stack();
        s.push(10);
        s.push(20);
        s.push(30); //add()
        S.O.P(s);
        S.O.P(s.pop()); //remove()
        S.O.P(s.pop());
        S.O.P(s.pop());
        //S.O.P(S.pop()); -> EmptyStackException
    }
}
```

O/P:-
[10, 20, 30]
30
20
10

Queue:-

* Queue is a predefined interface present in java.util package.
* Queue was introduced from JDK 1.5.
* Queue follows FIFO traversing order (First In First Out).
* The First object inside the queue is called 'head' & last object is called as 'tail'.

* The process of adding an object inside queue is called as 'Enqueue' & removing an object from the queue is called 'Dequeue'.

* The implementation classes of Queue are as follows:-
  - → Linked List
  - → Priority Queue

Priority Queue

* It is a predefined class present in java.util package.
* It was introduced from JDK 1.5.

Program:-

```
import java.util.PriorityQueue;
import java.util.Queue;
class Test{
    psvm(){
        Queue q=new PriorityQueue();
        q.add(10);
        q.add(20);
        q.add(30);
        S.O.P(q); //[10,20,30]
        S.O.P(q.poll()); //10
        S.O.P(q) //[20,30]
        q.clear();
        S.O.P("Is Empty?"+q.isEmpty()); //true
        S.O.P(q.poll()); //null
    }
}
```

O/P:-

[10,20,30]
10
[20,30]
Is Empty? true
null

	element()	peek()	remove()	poll()
Working	returns head element	returns head element	returns & removes head element	returns & removes head element
If Queue is Empty	NoSuchElement Exception	null	NoSuchElement Exception	null

Iterator:-

* Iterator is a predefined interface present in java.util package.
* Iterator was introduced from JDK 1.2.
* Iterator is uni-directional.

Important methods w.r.t Iterator

-> iterator():-

iterator() is used to return the iterable object's address

Syntax:- public Iterator<E> iterator();

-> hasNext():-

this method is used to check if there is an element in the next position or not.

Syntax: public boolean hasNext();

-> next():-

next() is used to return the object

Syntax:- public Element next();

Program:-

```
package org;
import java.util.*;
```

```
Class Solution{
    psvm(){
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(10);
        al.add(20);
        al.add(30);
        Iterator<Integer> i = al.iterator();
        while(i.hasNext()){
            S.O.P(i.next());
        }
        S.O.P("------");
        LinkedList<String> l = new LinkedList<String>();
        l.add("Apple");
        l.add("Banana");
        l.add("Mango");
        Iterator<String> itr = l.iterator();
        while(itr.hasNext()){
            S.O.P(itr.next());
        }
    }
}
```

O/P:- 10
20
30

Apple
Banana
Mango

Program for traversing using 3 different techniques

```
import java.util.*;
class Demo{
    P s v m (){
        Arraylist<Integer> l = new Arraylist<Integer>();
        l.add(10);
        l.add(20);
        for (int i=0; i<l.size(); i++)
        {   S.O.P (l.get(i));
        }
        for (int i : l)
        {   S.O.P (i);
        }
        Iterator<Integer> itr = l.iterator();
        while (itr.hasNext())
        {   S.O.P (itr.next());
        }
    }
}
```

O/P:-
10
20
10
20
10
20

3

Iterator
- Uni-directional
- Predefined Interface
- JDK 1.2
- java.util package

Methods
- → iterator()
- → hasNext()
- → next()

ListIterator
- Bi-Directional
- Predefined Interface
- JDK 1.2
- java.util package

Methods
- → listIterator()
- → hasPrevious()
- → previous()

Set:-

* Set is a pre-defined interface present in java.util package.
* Set was introduced from JDK 1.2.
* The implementation classes of set interface are as follows
  - → HashSet()
  - → LinkedHashSet()
  - → TreeSet()

```
Set s = new HashSet()
        new LinkedHashSet()
        new TreeSet()
```

Generalization/Upcasting

Specifications of Set interface

→ Insertion order is not maintained. except LinkedHashSet.
→ Duplication of objects is not allowed.
→ Set is not indexed based.
→ Null values can be stored.

Note:- We cannot use get() on set interface because get() will return the object based on index position & set is not index based. Therefore to traverse the objects from set interface we use iterators or enhanced for loop.

HashSet:-

* Hash Set is a pre-defined class present in java.util package.
* Hash Set was introduced from JDK 1.2.
* The Initial Capacity of Hash Set is 16
* The Fill ratio/Load Factor is 0.75 or 75%
* The underlined data structure of Hash Set is Hash Table.

Hash Table

key	value
Hash code of object	Object itself

Constructors present in HashSet

→ HashSet()
→ HashSet(int initialCapacity)
→ HashSet(int initialCapacity, float fillratio)
→ HashSet(Collection c)

Program:-

```
import java.util.HashSet;
class Demo{
    p s v m ( ){
        HashSet hs= new HashSet();
        hs.add(10);
        hs.add(null);
        hs.add(10);
        hs.add(true);
        hs.add("java");
        S.O.P(hs.size());
        S.O.P("-----");
        for(Object obj:hs){
            S.O.P(obj);
        }
    }
}
```

O/P:-

```
4
-------
null
java
10
true
```

LinkedHashSet:-

* LinkedHashSet is a pre-defined class present in java.util package.
* LinkedHashSet was introduced from JDK 1.4

* HashSet & LinkedHashSet are both similar but the major diff is HashSet doesnot maintain insertion order whereas LinkedHashSet maintains insertion order.

Program:-

```
import java.util.LinkedHashSet;
class Example{
    psvm(){
        LinkedHashSet<Integer> lhs = new LinkedHashSet<Integer>();
        lhs.add(10);
        lhs.add(56);
        lhs.add(23);
        lhs.add(23);
        for(int i: lhs){
            S.O.P(i);
        }
    }
}
```

O/P:-

```
10
56
23
```

Difference b/w List & Set:-

List	Set
* Insertion order is maintained	* Insertion order not maintained
* Duplicates are allowed	* Duplicates are not allowed
* List is index based	* Set is not index based.

Note:- HashSet & LinkedHashSet implements the following interfaces

→ Set
→ Cloneable
→ Serializable

10) Why Set interface does not allow duplicates

(or)

How set maintains uniqueness.

Internally hashCode() & equals() are overridden in all implementation classes, therefore set doesnot allow duplicates.

TreeSet :-

* TreeSet is a pre-defined implementation class of the set present in java.util package
* TreeSet was introduced from JDK 1.2
* TreeSet always follows natural sorting order (ascending order)

Program:-

```
import java.util.TreeSet;
class MainClass{
    p s v m ( ){
        TreeSet <Integer> t = new TreeSet<Integer>();
        t.add(30);
        t.add(40);
        t.add(20);
        t.add(30);
        t.add(50);
        t.add(10);
        for(int i:t){
            S.o.P(i);
        }
    }
}
```

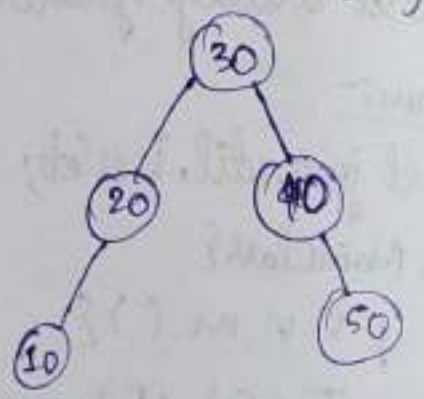
O/P:-

```
10
20
30
40
50
```

Working of TreeSet:-

* When u add an object into TreeSet Internally compareTo() gets called whose return type is "int"
* compareTo() returns '+1' when it is greater than (>)
  '-1' when it is lesser than (<)
  '0' when it is equal to (==).
* The underlying datastructure of preset is "binary tree"
* The traversing order of TreeSet Left node → Parent node & Right node

① t.add(30) → Set it as root node



② t.add(40);

40.compareTo(30)

40>30	40-30
+1	+10

RHS

③ t.add(20);

20.compareTo(30)

20<30	20-30
-1	-10

LHS

④ t.add(30);

30.compareTo(30)

30==30	30-30
0	0

Discard

⑤ t.add(50);

50.compareTo(30)

50>30	50-30
+1	+20

RHS

⑥ t.add(10);

10.compareTo(30)

10<30	10-30
-1	-20

10.compareTo(20)

10<20	10-20
-1	-10

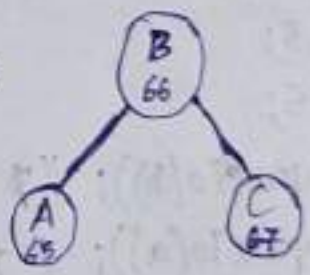
Note:- TreeSet allows only homogeneous objects

Ex Pgm:-

```
import java.util.TreeSet;
class MainClass{
    p s v m ( ){
        TreeSet<String> t = new TreeSet<String>();
        t.add("B");
        t.add("C");
        t.add("A");
        for (String i : t){
            S.O.P(i);
        }
    }
}
```

O/P:-
A
B
C

① t.add("B") -> Set it as root node



② t.add("C");
"C".compareTo("B");
67 66

67>66 | 67-66 RHS
+1 | +1

③ t.add("A")
"A".compareTo("B");
65 66

65<66 | 65-66 LHS
-1 | -1

Note:- * When u add an integer object into TreeSet it calls compareTo() present in Integer class, likewise for all the other objects.
* compareTo() is present in all Wrapper classes and String class

Program:

```
import java.util.*;
class Runner{
    psvm(){
        String x="A";
        String y="B";
        S.O.P(x.compareTo(y));  //-1
        S.O.P(y.compareTo(x));  //1
        S.O.P(x.compareTo(x));  //0

        Integer i=10;
        Integer j=20;
        S.O.P(i.compareTo(j));  //-1
        S.O.P(j.compareTo(i));  //1
        S.O.P(j.compareTo(j));  //0
        S.O.P("-----");
        Double a=3.5;
        Double b=2.5;
        S.O.P(a.compareTo(b));  //1
        S.O.P(b.compareTo(a));  //-1
        S.O.P(a.compareTo(a));  //0
    }
}
```

WAJP to find the largest element inside ArrayList

```
import java.util.ArrayList;
class Largest{
    psvm(){
        ArrayList<Integer> l=new ArrayList<Integer>();
        l.add(10);
        l.add(30);
        l.add(25);
        l.add(40);
```

```
l.add(15);
S.O.P(l);
int max = l.get(0);
for(int i=1; i<l.size(); i++){
    if(l.get(i) > max){
        max = l.get(i);
    }
}
S.O.P("MAX:"+max);
int large = l.get(0);
for(int i : l){
    if(i > large){
        large = i;
    }
}
S.O.P("MAX:"+large);
}
```

O/P:-

[10, 20, 25, 40, 15]

MAX: 60

MAX: 60

Tracing:-

l | 10 | 40 | 20 | 30 | 10 |

```
-> int max = l.get(0);
for(int i=1; i<l.size(); i++){
    if(l.get(i) > max){
        max = l.get(i);
    }
}
S.O.P(max);
```

max = 50

WAJP to store 4 emp objects into ArrayList & print the name of the employee with a highest salary.

```
import java.util.*;
class HighestSalary{
    p s v m ( ) {
class Employee{
    String name;
    double salary;
    Employee(String name, double salary){
```

```
        this.name = name;
        this.salary = salary;
    }
    @Override
    public String toString(){
        return "Salary of "+name+" is Rs. "+ salary;
    }
}
class MainClass{
    p s v m ( ) {
        Arraylist<Employee> al = new Arraylist <Employee> ();
        l.add (new Employee ("Tom", 1.5));
        l.add (new Employee ("King", 3.5));
        l.add (new Employee ("Smith", 7.5));
        l.add (new Employee ("Jack", 5.5));
        for(Employee emp: l){
            S.O.P(emp);
        }
        S.O.P ("-------");
        Employee max = l.get(0);
        for(int i=1; i<l.size(); i++){
            if(l.get(i).salary > max.salary){
                max = l.get(i);
            }
        }
        S.O.P ("MAX:"+max);
    }
}
```

O/P:

```
Salary of Tom is 1.5
King is 3.5
Smith is 7.5
Jack is 5.5
-------
MAX: Salary of Smith is 7.5
```

Comparable :-

* Comparable is used to compare the objects & sort them either in ascending order or descending order.
* Comparable has an abstract method called as CompareTo().
* Comparable is a pre-defined interface present in java.lang package
* Comparable ~~is used~~ was introduced from JDK 1.2
* Syntax:-

```
public int compareTo(Object obj);
```

Already existing object

Element

+1 -1 0
> < ==

* Comparable is used for default sorting & mutual comparision.

Rules for using comparable interface or Rules to make the objects as comparable in nature

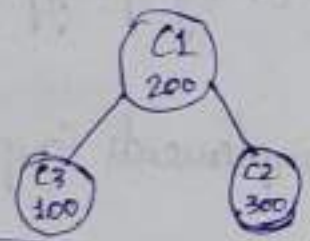
1) class should implement Comparable interface.
2) Specify generics indicating which objects should be compared.
3) Override compareTo() by specifying the business logic of comparision.

Ex Program:-

```
public class Car implements Comparable<Car> {
    int cost;
    Car (int cost){
        this.cost = cost;
    }
    @Override
    public String toString(){
        return "Cost:"+cost;
    }
```

```
@Override
public int compareTo(Car c){
    return this.cost - c.cost;
}
}
class SortCar{
    public static void main(String[] args){
        Car c1 = new Car(200);
        Car c2 = new Car(300);
        Car c1 = new Car(100);
        TreeSet<Car> t = new TreeSet<Car>();
        t.add(c1);
        t.add(c2);
        t.add(c3);
        for(Car c : t){
            S.O.P(c);
        }
    }
}
```

O/P:-
Cost : 100
Cost : 200
Cost : 300



Tracing:-

① t.add(c1) → get c1 object as root node

② t.add(c2)

c2.compareTo(c1)
this, c

return this.cost - c.cost

c2.cost - c1.cost

300 > 200 | 200 - 200 → RHS
+100 | +1

③ t.add(c3)

c3.compareTo(c1)
this, c

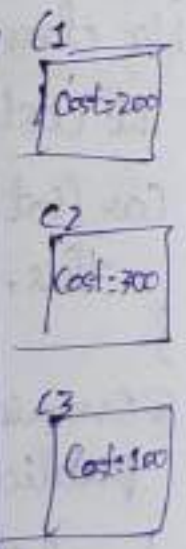
return this.cost - c.cost

c3.cost - c1.cost

100 < 200 | 100 - 200 ⇒ LHS
-100 | -1

```
@Override
public int compareTo(Car c)
{
    return this.cost - c.cost;
}
```

Already existing object



Ex②:-

```
class Student implements Comparable<Student>{
    int age;
    Student(int age){
        this.age=age;
    }
    @Override
    public String toString(){
        return "Age is"+age;
    }
    @Override //Descending Order
    public int compareTo(Student s){
        return s.age - this.age; // return this.age - s.age [Ascending order]
                                 // Already Existing Object Age - Object To be Inserted Age;
    }
}
class SortStudent{
    psvm(){
        Student s1=new Student(21);
        Student s2=new Student(22);
        Student s3=new Student(20);
        TreeSet<Student> t=new TreeSet<Student>();
        t.add(s1);
        t.add(s2);
        t.add(s3);
        for(Student s:t){
            S.O.P(s);
        }
    }
}
```

O/P:-

Age is 22
Age is 21
Age is 20

Tracing :-

① t.add(S1);

② t.add(S2);

S2.compareTo(S1);

this s

return s.age - this.age

s1.age - s2.age

21>22 | 21-22 ⇒ LHS

-1 | -1

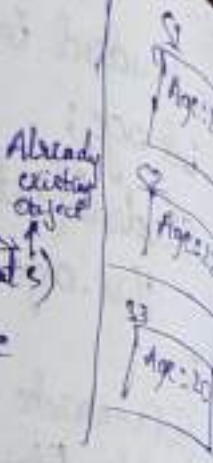
@Override

public int compareTo(Student s)

{ return s.age - this.age

}

Already existing object



③ t.add(S3);

S3.compareTo(S1);

this s

return s.age - this.age

S1.age - S3.age

21<20 | 21-20 ⇒ RHS

+1 | +1

Ex③ Program:-

```
class Employee implements Comparable<Employee>{
    String name;
    Double salary; // Wrapper class Declaration
    Employee (String name, double salary){
        this.name = name;
        this.salary = salary;
    }
    @Override
    public String toString(){
        return "Name: "+name+" Salary: "+salary;
    }
    @Override
    public int compareTo(Employee e){
        return this.name.compareTo(e.name);
    }
```

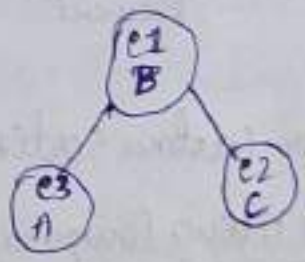
//return this.name.compareTo(e.name); → name comparison → ascending order
//return e.name.compareTo(this.name); → name comparison → descending order
//return this.salary.compareTo(e.salary); → salary comparison → ascending order
//return e.salary.compareTo(this.salary); → salary comparison → descending order

```
class SortEmployee{
    p s v m () {
        TreeSet<Employee> e = new TreeSet<Employee> ();
        t.add(new Employee("B", 3.5)); //e1
        t.add(new Employee("C", 1.5)); //e2
        t.add(new Employee("A", 2.5)); //e3
        for(Employee emp : t){
            s.o.p(emp);
        }
    }
}
```

O/P:-

Name: A Salary: 2.5
Name: B Salary: 3.5
Name: C Salary: 1.5

Tracing:-



① t.add(e1);

② t.add(e2);

e2.compareTo(e1)
this → e

this.name.compareTo(e.name)
e2.name.compareTo(e1.name)
"C".compareTo("B")
67 → 66

67 > 66 | 67 - 66 ⇒ RHS
+1 | +1

③ t.add(e3);

e3.compareTo(e1)
this → e

this.name.compareTo(e.name)
e3.name.compareTo(e1.name)
"A".compareTo("B")
65 → 66

65 < 66 | 65 - 66 ⇒ RHS
-1 | -1

```
@Override
public int compareTo(Employee e)
{
    return this.name.compareTo(e.name);
}
```

Note:- All Wrapper classes & String class implements comparable interface and has overridden compareTo(), therefore to compare String objects we have compareTo() in String class, likewise for the other types.

Comparator:-

* Comparator is a pre-defined interface present in java.util package.
* It was introduced from JDK 1.2.
* Comparator is also used for comparing objects and sorting them either in ascending order or descending order.
* Comparator has an abstract method called as Compare().
* Syntax:-

```
                                  object to be inserted      Already existing object
public int compare( E e1 , E e2 )
                    ↓            ↓
                  Element      Element
```

```
+1 -1 0
> < ==
```

* Comparator is used for custom sorting & unmutual comparison.

Rules for comparing objects using Comparator interface

1) Create a class which implements Comparator interface and import it from java.util package.
2) Specify generics indicating the objects to be compared.
3) Override compare() by specifying the logic of comparison.
4) Create an object of the class where the sorting logic is defined and pass that object to the constructor of TreeSet.

Program:

```
class Student {
    int id;
    int age;
    Student (int id, int age){
      this.id=id;
      this.age=age;
    }
    @Override
    public String toString(){
      return "Id:"+id+"\t Age:"+age;
    }
}
class SortStudentById implements Comparator<Student>{
    @Override
    public int compare (Student x, Student y){
      return x.id - y.id; // ascending order
    }
}
//return y.id - x.id; //descending order
class SortStudentByAge implements Comparator<Student>{
    @Override
    public int compare (Student x, Student y){
      return y.age - x.age; //descending order
    }
}   //return x.age-y.age; //ascending order
class SortStudent{
    p s v m ( ) {
        Student s1= new Student (200, 23);
        Student s2= new Student (300, 21);
        Student s3= new Student (100, 22);
```

```
/*SortStudentById id = new SortStudentById();
  TreeSet<Student> t = new TreeSet<Student>(id); */

// TreeSet<Student> t = new TreeSet<Student>(new SortStudentById());

TreeSet<Student> t = new TreeSet<Student>(new SortStudentByAge());
t.add(s1);
t.add(s2);
t.add(s3);
for(Student std : t){
    S.O.P(std);
}
}
```

O/P :-

Id : 200 Age : 23
Id : 100 Age : 22
Id : 300 Age : 21

}

Tracing (To Sorting Student object based on id in ascending order)

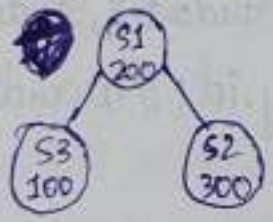
① t.add(s1);

② t.add(s2);
↓
compare(x, y)
↓ ↓
s2 s1

x.id - y.id
s2.id - s1.id
300 - 200 | 300-200 ⇒ RHS
+100 | +1

③ t.add(s3);
↓
compare(x, y)
↓ ↓
s3 s1

x.id - y.id
s3.id - s1.id
100 - 200 | 100-200 ⇒ LHS
-100 | -1



@Override
public int compare(Student x, Student y)
{
    return x.id - y.id;
}

Program:

```
class Hotel{
    String name;
    Double ratings; //Wrapper class declaration to call compareTo() later
    Hotel(String name, double ratings){
        this.name = name;
        this.ratings = ratings;
    }
    @Override
    public String toString(){
        return "Ratings of "+name+" Hotel is "+ratings;
    }
}
class SortHotelByName implements Comparator<Hotel>{
    @Override
    public int compare(Hotel x, Hotel y){
        return x.name.compareTo(y.name); //ascending order
    }
}// return y.name.compareTo(x.name); -> descending order
class SortHotelByRatings implements Comparator<Hotel>{
    @Override
    public int compare(Hotel x, Hotel y){
        return y.ratings.compareTo(x.ratings); //descending order
    }
}//return x.ratings.compareTo(y.ratings); -> ascending order
class SortHotel{
    psvm(){
        TreeSet<Hotel> t = new TreeSet<Hotel>(new SortHotelByRatings());
        t.add(new Hotel("Taj", 4.8));
```

```
        t.add(new Hotel("ITC", 4.7));
        t.add(new Hotel("Empire", 3.8));
        t.add(new Hotel("Brahmins", 4.1));
        for(Hotel h : t){
            S.O.P(h);
        }
    }
}
```

O/P:-
Ratings of Taj Hotel is
Ratings of ITC Hotel is
Ratings of Brahmin Hotel
Ratings of Empire Hotel is

Diff b/w Comparable & Comparator

Comparable	Comparator
* Pre defined interface present in "java.lang" package & introduced from JDK 1.2	* Pre defined interface present in "java.util" package & introduced from JDK 1.2.
* public int compareTo(Element e);	* public int compare(E e1, E e2);
* Default sorting	* Custom sorting
* Mutual Comparison.	* Unmutual Comparison.
* Java source code & the sorting logic is present in the same class. (Dependent on each other).	* Java source code & the sorting logic is present in seperate class. (Independent of each other).

Q) Write a java program to compare mobile objects based on cost descending order using comparable interface & ArrayList class.

Sol:
```
class Mobile implements Comparable<Mobile>{
    int cost;
    Mobile(int cost){
```

```
    this.cost=cost;
  }
  @Override
  public String toString(){
    return "Cost: "+cost;
  }
  @Override
  public int compareTo(Mobile m){
    return m.cost - this.cost;
  }
}
import java.util.ArrayList;
class SortMobileUsingCost{
  psvm (){
    ArrayList<Mobile> al=new ArrayList<Mobile>();
    al.add(new Mobile(20000));
    al.add(new Mobile(25000));
    al.add(new Mobile(18000));
    Collections.sort(al);
    for(Mobile m: al){
      S.O.P(m);
    }
  }
}
```

O/P:-
Cost: 25000
Cost: 20000
Cost: 18000

```
class Beer{
  String brand;
  int cost;
  Beer(String brand, int cost){
    this.brand=brand;
```

```
    this.cost = cost;
  }
  @Override
  public String toString(){
    return "Cost of " + brand + "beer is" + cost;
  }
}import java.util.Comparator;
class SortBeerBybrand implements Comparator<Beer> {
  @Override
  public int compare(Beer x, Beer y){
    return x.brand.compareTo(y.brand);
  }
}import java.util.Comparator;
class SortBeerByCost implements Comparator<Beer>{
  @Override
  public int compare(Beer x, Beer y){
    return y.cost - x.cost;
  }
}import java.util.*;
class SortBeer{
  p s v m ( ){
    ArrayList<Beer> listofBeerObjects = new ArrayList<Beer>();
    listofBeerObjects.add(new Beer("Tuborg", 190));
      — " —          (new Beer("KF", 160));
      — " —          (new Beer("Budweiser", 220));

    Collections.sort(listofBeerObjects, new SortBeerByCost());
    //Collections.sort(listofBeerObjects, new SortBeerBybrand());
```

```
for(Beer b : listofBeerObjects){
    S.O.P(b);
  }
}
```

O/P:

Cost of Budweiser beer is 220
Cost of Tuborg beer is 190
Cost of KF beer is 160

}

Q) Write a Java program to store 4 Employee objects into a collection, print employee details & find the names of the employee with the highest salary & lowest salary.

Sol:

```
class Employee implements Comparable<Employee>
{
    String name;
    int Salary;
    Employee(String name, int salary){
      this.name = name;
      this.salary = salary;
    }
    @Override
    public String toString(){
      return "Salary of " + name + " is " + salary;}
    @Override
    public int compareTo(Employee e){
      return this.salary - e.salary;}
}
class SortEmployee
{
    public static void main(String[] args)
    {
      ArrayList<Employee> al = new ArrayList<Employee>();
      al.add(new Employee("Tom", 25000));
      al.add(new Employee("Jack", 30000));
      al.add(new Employee("Smith", 20000));
      al.add(new Employee("King", 35000));
```

```
S.O.P(al);
int max = al.get[0];
for(int i=1; i<al.size(); i++)
{
    if(l.get[i].salary > max.salary)
    {
        max = l.get[i];
    }
}
Collections.sort(al);
S.O.P("Lowest salary -> " + al.get(0));
S.O.P("Highest salary -> " + al.get(al.size()-1));
}
}
```

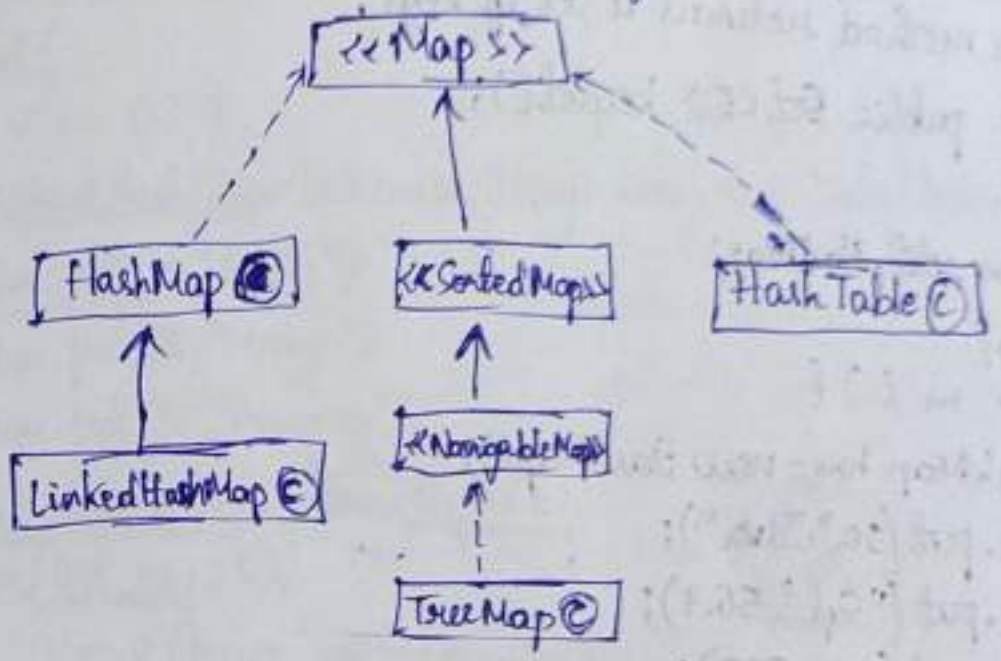
O/P:-

Lowest salary → Salary of Smith is 20000
Highest salary → Salary of King is 35000

Map:-

* Map is a pre-defined interface present in <u>java.util package</u>.
* Map was introduced from <u>JDK 1.2</u>.
* Map is a part of collection framework but map doesnot extends Collection interface.
* Map stores the objects in the form of key & value pair, wherein the keys cannot be duplicated but values can be duplicated.
* One key & value together is called as 'Entry' therefore map is a collection of entries.
* The implementation classes of map interface are as follows:
  - → HashMap
  - → LinkedHashMap
  - → TreeMap
  - → HashTable

Map Hierarchy



Important methods present in Map interface

1) **put():-**
put() is used to add an entry into the Map.

2) **get():-**
get() is used to return the value based on the key.

3) **containKey():-**
containKey() is used to check if the key is present or not.

4) **containsValue():-**
containsValue() is used to check if the value is present or not.

5) **remove():-**
remove() is used to remove the value based on the key specified.

6) **size():-**
size() is used calculate the number of entries.

7) **clear():-**
clear() is used to remove all the entries from the map.

8) **isEmpty():-**
isEmpty() is used check if the entry is empty or not.

9) ~~public Set<E>~~ keySet():-

This method returns a set of keys.

Syntax:- public Set<E> keySet();

Ex Program:-

```
import java.util.HashMap;
class Demo{
    p s v m ( ){
        HashMap hm=new HashMap();
        hm.put(10,"Java");
        hm.put("Sql",56.7);
        hm.put(1.4,555);
        S.O.P(hm); // {1.4=555, 10=Java, Sql=56.7}
        S.O.P(hm.get("Sql")); //56.7
        S.O.P(hm.containsKey(10)); //true
        S.O.P(hm.containsValue("java")); //false
        S.O.P(hm.size()); //3
        S.O.P("--------");
        S.O.P(hm); //{1.4=555, 10=Java, Sql=56.7}
        hm.remove(10);
        S.O.P(hm); // {1.4=555, Sql=56.7}
        S.O.P("--------");
        S.O.P(hm.isEmpty()); //false
        hm.clear();
        S.O.P(hm.isEmpty()); //true
    }
}
```

Program:-

```
import java.util.*;
class Test{
  p s v m ( ){
    LinkedHashMap<Integer,String> lhm = new LinkedHashMap<Integer,String>();
    lhm.put(10,"Apple");
    lhm.put(20,"Mango");
    lhm.put(30,"Banana");
    Set<Integer> s = lhm.keySet();
    for(int key : s){
      S.O.P(key+" -> "+lhm.get(key));
    }
    S.O.P("-------");
    TreeMap<Double, Integer> map = new TreeMap<Double,Integer>();
    map.put(2.2,44);
    map.put(3.1,54);
    map.put(1.7,64);
    Set<Double> setOfKeys = map.keySet();
    for(double key : setOfKeys){
      S.O.P(key+" -> "+map.get(key));
    }
  }
}
```

Op:-
10 -> Apple
20 -> Mango
30 -> Banana

1.7 -> 64
2.2 -> 44
3.1 -> 54

Tracing:-

① Set<Integer> s = lhm.keySet()
for (int key : s)
{
S.O.P(key+" -> "+lhm.get(key));
}

10 Apple
20 Mango
30 Banana

s [10 | 20 | 30]

lhm

key	value
10	Apple
20	Mango
30	Banana

① Map -> Traverse ✗

② Map -> Set of Keys -> Traverse
get()

Note:- * HashMap doesnot maintain insertion order.
* LinkedHashMap maintains insertion order.
* TreeMap follows natural sorting order (ascending order) based on Keys)

---

6) WAJP to store objects inside a linkedHashMap where in key is String indicating name & value is double indicating salary. Traverse the entries in the below format "Salary of Tom is Rs. 10.50"

```
import java.util.LinkedHashMap;
import java.util.Set;
class Demo{
    p s v m (){
        LinkedHashMap<String,Double> lhm = new LinkedHashMap
        <String, Double>();
        lhm.put("Tom", 10.50);
        lhm.put("King", 30.50);
        lhm.put("Smith", 20.50);
        Set<String> setOfNames = lhm.keySet();
        for(String key : setOfNames){
            S.O.P("Salary of" + key + " is Rs." + lhm.get(key));
        }
    }
}
```

O/P:-
Salary of Tom is Rs 10.50
Salary of King is Rs 30.50
Salary of Smith is Rs 20.50

Note: When we have a duplication of key the value gets overridden

```
Class MainClass{
    p s v m () {
        LinkedHashMap map = new LinkedHashMap();
        map.put(10,"Sony");
        map.put(20,"Boat");
        S.O.P(map);
        map.put(10,"JBL");
        s.O.P(map);
    }
}
```

O/P:-
{10=Sony, 20=Boat}
{10=JBL, 20=Boat}

Q) WAJP by following below scenarios

i) Create a class called as Student

ii) Declare 2 attributes called as Name & subject

iii) Initialize those attributes with the help of a constructor

iv) Create another call-class called as Solution, under main method create an instance of LinkedHashMap & store ~~emp~~ student objects as the key & marks as the values. [Hint: key is Student & value is Integer].

v) Traverse the entries in the below format.

[Tom has scored 65 marks in Java.]

Sol
```
class Student
{
    String name;
    String subject;
    Student (String name, String subject){
        this.name=name;
        this.subject=subject;
    }
}
```

```
class Solution{
    psvm () {
        Student s1 = new Student("Jack","Python");
        LinkedHashMap<Student,Integer> map = new LinkedHashMap
            <Student,Integer>();
            map.put(new Student("John","Java"),65);
            map.put(new Student("Tom","JavaScript",75);
            ~~map.put(new Student("Smith","Web",85);~~
            map.put(s1,85);
            Set<Student> setOfKeys = map.keySet();
            for(Student key : setOfKeys){
                S.O.P(key.name+" has scored "+map.get(key)+" marks in "+key.subject)
            }
    }
}
```

op:-

Jack has scored 85 marks in Python
John has scored 65 marks in Java
Tom has scored 75 marks in JavaScript

HashMap:-

HashMap is a pre-defined class present in java.util package introduced from JDK 1.2

HashMap doesnot maintain insertion order

LinkedHashMap:-

LinkedHashMap is a pre-defined class present in java.util package introduced from JDK 1.4

LinkedHashMap maintains insertion order.

TreeMap :-

TreeMap is a predefined class present in java.util package introduced from JDK 1.2

TreeMap maintains sorted order (ascending order).

HashTable :-

HashTable is a pre-defined class present in java.util package introduced from JDK 1.0 (Legacy Collections).

HashMap & HashTable are both similar but HashTable is ThreadSafe where HashMap is not ThreadSafe.

Multi Threading

Thread:- * A thread is a small light-weight sub process.

* In other words, if we want to perform or achieve a task, we divide the task to multiple sub tasks, therefore one small sub-task is called as a thread.
* Java is a multi-threaded language (multiple threads are running simultaneously when we execute a single java program.
* In java's perspective thread is a pre-defined class present in java.lang package.

Properties of a Thread:-

1) Thread id:- Thread id is used to identify a thread uniquely.
2) Thread name:- Thread name is used to identify a thread easily.

Note:- Thread id cannot be duplicated but thread name can be duplicated.

3) Thread priority:- Thread priority is used to decide, the execution of a thread.

* Thread priority ranges from 1 to 10
  - → 1 being lowest/least priority
  - → 5 being default priority
  - → 10 being the highest priority

Note:- Thread id, thread name & thread priority are private data members in thread class, therefore we can access outside the class with the

help of getter() & setter().

Important methods present in Thread class

1) getId():-
   getId() is used to return the id of a thread.

2) getName():-
   getName() is used to return the name of a thread.

3) setName():-
   setName() is used to set the name for thread.

4) getPriority():-
   getPriority() is used to return the priority of a thread.

5) setPriority():-
   setPriority() is used to set the priority for a thread.

6) start():-

7) run():-

8) sleep():-

9) currentThread():-

10) stop():-

Program:-

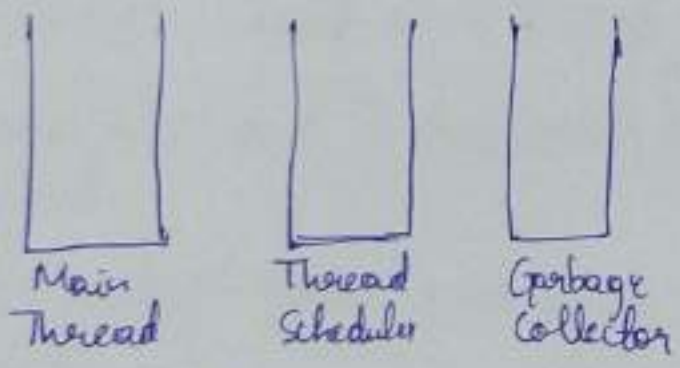
```
class Demo{
    P S V m ()
    {
        Thread t = new Thread();
        S.O.P(t.getId());          //10
        S.O.P(t.getName());        //Thread-0
        S.O.P(t.getPriority());    //5
```

```
        S.O.P("------");
        t.setName("GuldeThread");
        t.setPriority(2);
        S.O.P(t.getId());          //10
        S.O.P(t.getName());        // GuldeThread
        S.O.P(t.getPriority());    //2
    }
}
```

Pre-Defined Threads:-

When we execute a single java program, 3 threads will get executed simultaneously.

1) Main Thread
2) Thread Scheduler
3) Garbage Collector



* Main Thread is where in the execution of a program happens (the entry point of the java program is the main thread or main method).
* Thread Scheduler is used to manage the threads (thread scheduler will decide when the thread needs to be executed.

* Garbage Collector:- All the dereferenced objects are automatically deleted from the memory with the help of garbage collector.

Constructors present in Thread class

1) Thread ()
2) Thread (String name)
3) Thread (Runnable obj)
4) Thread (Runnable obj, String name)

User-Defined Threads or Custom Threads

In java it is possible to create user-defined thread in 2 ways

→ Thread class

→ Runnable interface.

Rules for creating a thread using thread class.

1) class should inherit thread class
2) Override run() by specifying the task which needs to be performed by the thread.

   Syntax:- public void run()
3) Invoke the start().

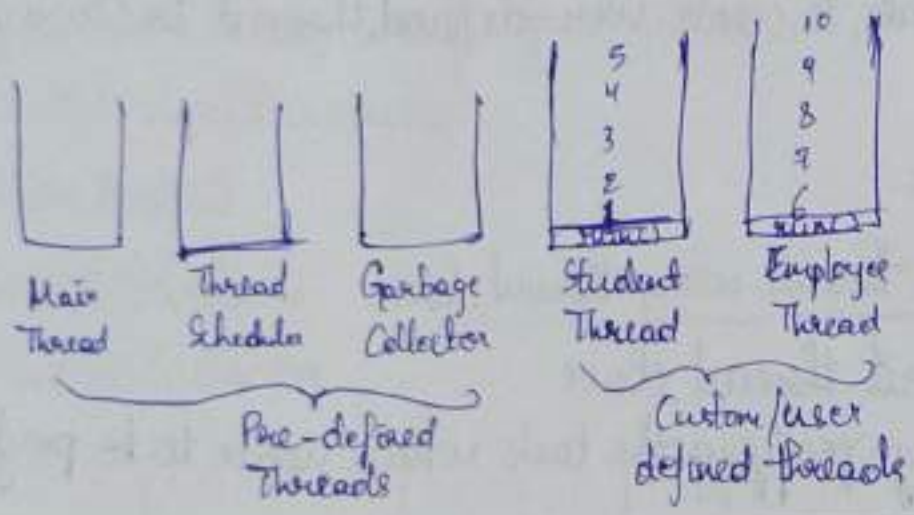
Program:-

```
class Student extends Thread
{
    @Override
    public void run(){
        for(int i=1; i<=5; i++){
            S.O.P("i:"+i);
            try{ Thread.sleep(1000); }
            catch(Exception e){ e.printStackTrace(); }
        }
    }
}
```

```
class Employee extends Thread
{
    @Override
    public void run(){
        for(int j=6; j<=10; j++){
            S.O.P("j:"+j);
            try{ Thread.sleep(1000); }
            catch(Exception e){ e.printStackTrace(); }
        }
    }
}
```

```
Class Solution {
    p s v m () {
        Employee emp = new Employee();
        emp.setName("Employee thread");
        emp.start();
        Student std = new Student();
        std.setName("Student thread");
        std.start();
    }
}
```

O/P:-
i=1
j=6
i=2
i=3
j=7
j=8
i=4
j=9
i=5
j=10



Note:-
* sleep() is used to pause the execution for a specified millisecond
* run() is overridden to perform the task as a thread

Why do we call start() & not run()?
⇒ When we invoke the run() the execution will happen on the existing stack where in a thread has not got created, therefore we invoke start() where in start() creates a new stack and invokes the run() on the newly created stack.

Q) WAJP to create a custom thread whose task is to print even number from 1 to 10.

```
class EvenNo extends Thread{
  @Override
  public void run(){
    for(int i=1; i<=10; i++){  or for(int i=2; i<=10; i=i+2)
      if(i%2==0){
        S.O.P(i);
      }
    }
  }
}
class Runner(){
  p s v m(){
    EvenNo eno = new EvenNo();
    eno.start();
  }
}
```

o/p:-
2
4
6
8
10

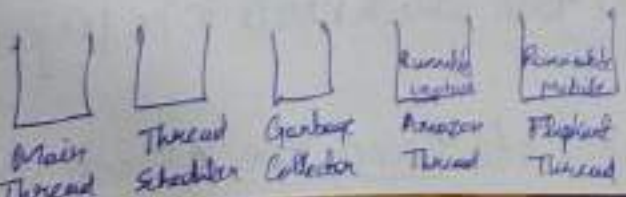
```
//Default Name is provided
Thread t1 = new Thread();
t1.setName("dinga"); //setting the name using setter method
S.O.P(t1.getName());
//setting the name of the thread using constructor
Thread t2 = new Thread("guldu");
S.O.P(t2.getName());
```

Rules for creating a thread using Runnable interface

1) class should implement Runnable interface
2) Override run() by defining the task which needs to be performed by thread.
3) Convert the Runnable type of object into thread type 4) invoke start()

Program:-

```
class Amazon implements Runnable {
    @Override
    public void run() {
        S.O.P ("Purchasing laptop");
    }
}
class Flipkart implements Runnable {
    @Override
    public void run() {
        S.O.P ("Purchasing Mobile");
    }
}
class Solution {
    p s v m () {
        Amazon a = new Amazon();
        Thread t1 = new Thread (a);
        t1.setName("Amazon Thread");
        S.O.P (t1.getName());
        t1.start();
        Flipkart f = new Flipkart();
        Thread t2 = new Thread (f, "Flipkart Thread");
        S.O.P (t2.getName());
        t2.start();
    }
}
```



O/P:-
Amazon Thread
Purchasing laptop
Flipkart Thread
Purchasing Mobile

```
class Uber extends Thread {
    @Override
    public void run (){
        S.O.P ("Travelling to Basavanagudi");
    }
}
class PayTM implements Runnable {
    @Override
    public void run(){
        S.O.P ("Paying through Paytm");
    }
}
class MainClass {
    p s v m ( ) {
        Uber u = new Uber();
        u.start();
        PayTM p = new PayTM();
        Thread t1 = new Thread (p);
        t1.start ();
    }
}
```

O/P:-

Travelling to Basavanagudi
Paying through Paytm

or new Uber().start();

or ~~new Thread(p).start();~~ new Thread(new PayTM()).start();

-A Relationship:-

Ex:-

```
class Person {
    String name = "Tom";
    Mobile mobile = new Mobile();
    static int id = 101;
    static Bike bike = new Bike();
```

```
}
class Bike{
    String brand = "Honda";
}
class Mobile{
    String brand = "Apple";
}
class Solution{
    p s v m ()
    {
        Person person = new Person ();
        S.O.P("Name:" + person.name);
        S.O.P("Brand:" + person.mobile.brand);
        S.O.P("--------");
        S.O.P("Id:" + Person.id);
        S.O.P("Brand:" + Person.bike.brand);
    }
}
```

O/P:-

```
Name: Tom
Brand: Apple
---------
Id: 101
Brand: Honda
```

Ex:- (Without using constructor)

```
class College{
    String name = "BIT";
    Student student = new Student();
  XLaptop laptop = new Laptop();
}
class Student{
    int age = 23;
    int id = 101;
    String name = "Praveen";
    Laptop laptop = new Laptop();
}
```

```
class Laptop {
    String brand = "HP";
    int cost = 40000;
}
class Solution {
    p s v m ( ) {
        College college = new College();
        S.O.P ("College Name: " + college.name);
        S.O.P ("Student Age: " + College.student.age);
        S.O.P ("Student id: " + College.student.id);
        S.O.P ("Student Name: " + College.student.name);
        S.O.P ("Laptop Brand: " + college.student.laptop.brand);
        S.O.P ("Laptop Cost: " + college.student.laptop.cost);
    }
}
```

Ex:- (Using Constructor)

```
class College {
    String name; //String name = "Jspiders";
    Student student; //Student student = new Student("Jack");
    College(String name, Student student){
        this.name = name;
        this.student = student;
    }
}
class Student {
    String name;
    Student(String name){
        this.name = name;
```

o/p:-

College name: Jspiders
Student name: Jack

```
Class Solution{
    p s v m ( ){
        College college = new College("Jspiders", new Student("Jack"));
        S.O.P("College name: "+college.name);
        S.O.P("Student name: "+college.student.name);
    }
}
```

Thread Safe / Synchronization

→ If multiple threads access an object or shared resource at a time it will lead to faulty results / erroneous results where in the thread is not safe, therefore it is sometimes necessary to make the thread safe.

→ Thread safe is a process where in, if one thread is accessing an object or shared resource at a time, the remaining threads will be in waiting state, this process is called as Synchronization or thread safe.

→ Thread safe is achieved with the help of "synchronized" keyword or "synchronized" block.

Rules for achieving Thread safe

1) Identify the object or shared resource and threads and then create it as seperate classes.
2) Achieve Has-A Relationship between the shared resource and the threads.
3) Create a thread, override run() & invoke the start()

Program:-

```
package threadsafe;
class Trainer {
    synchronized void askQuestion(String name){
        S.O.P (name+ "is asking a Question?");
        try {
            Thread.sleep(2000);
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }
        S.O.P(name+" has got the Answer \n------");
    }

}
class Student extends Thread {
    String name;
    Trainer trainer; // Trainer trainer= new Trainer();
    Student (String name, Trainer trainer){
        this.name=name;
        this.trainer=trainer;
    }
    @Override
    public void run(){
        trainer.askQuestion(name);
    }

}
class Solution {
    P S V M ( ) {
        Trainer trainer = new Trainer(); // Shared Resource
```

```
    Student s1 = new Student ("Tom", trainer);
    Student s2 = new Student ("Jack", trainer);
    Student s3 = new Student ("Alex", trainer);
    s1.start();
    s2.start();
    s3.start();
  }
}
```

O/P:-

```
Tom is asking a Question?
Tom has got the Answer
- - - - - - - -
Jack is asking a Question?
Jack has got the Answer
- - - - - - - -
Alex is asking a Question?
Alex has got the Answer
```

Thread Safe/Synchronization Continuation

Program:-

```
class Netflix {
    synchronized void watchMovie(String name){
        S.O.P(name+" is watching some movie");
        try{
            Thread.sleep(2000);
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        S.O.P(name+" is done watching the movie \n-----");
    }
}
class Person extends Thread {
    String name;
    Netflix netflix;
    Person(String name, Netflix netflix){
        this.name=name;
        this.netflix=netflix;
    }
```

```
@Override
public void run(){
    netflix.watchMovie(name);
}
}
class Runner{
    p s v m ( ){
        Netflix netflix = new Netflix();
        Person p1 = new Person("Tom", netflix);
        Person p2 = new Person("Jerry", netflix);
        p1.start();
        p2.start();
    }
}
```

O/P:-

Tom is watching some movie
Tom is done watching the movie
- - - - - - - -
Jerry is watching some movie
Jerry is done watching the movie
- - - - - - - -

Note:-

currentThread() is used to get the information about the currently executing thread.

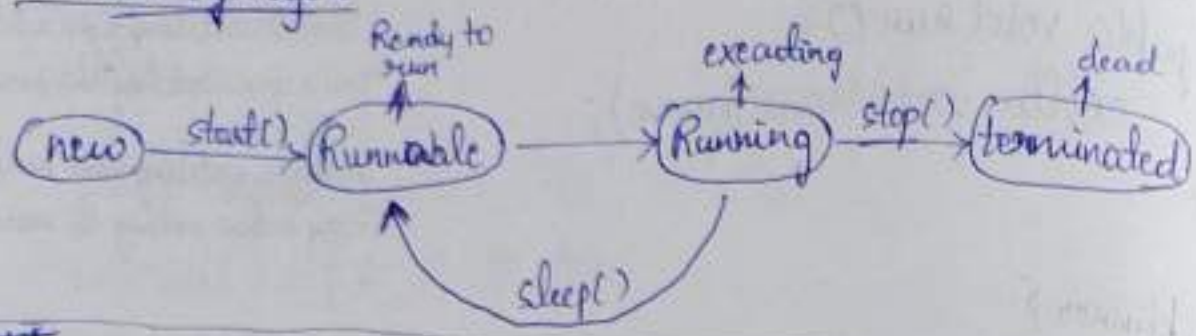
Ex:-

```
class Test{
    public static void main(String[] args)
    {
        Thread t = Thread.currentThread();
        S.O.P("Name:"+t.getName());
        S.O.P("Id:"+t.getId());
        S.O.P("Priority:"+t.getPriority());
    }
}
```

O/P:-

Name : main
Id : 1
Priority : 5

Thread Life Cycle



Q) How do you explicitly delete an object from the memory (Garbage Collection)?

Sol We can explicitly remove an object from the memory by invoking [gc()] which is present in 'System' class.

Rules for using explicit garbage collector:-

1) Re initialize the reference variable to null.
2) Invoke gc() present in System class

Syntax:-

```
System.gc();
```

Ex Program:-

```
class Student {
    psvm ( ) {
        Student s = new Student();
        s = null; //Object re-initialized to null (Eligible for GC)
        System.gc(); //Invoking the Garbage Collection
    }
}
```

Difference b/w final, finally and finalize.

→ final:-

* final is a keyword, which can be used with a variable, method and class.

* final variable cannot be re-initialized.
* final() cannot be overridden.
* final class cannot be inherited.

→ finally :-

* finally is a block, which gets executed always even if an exception occurs or not.

→ finalize() :-

* finalize is a method, which is used during garbage collection
* finalize() gets executed just before an object is garbage collected.
* finalize() is present in Object class

Syntax :- protected void finalize()

Ex Program :-

```
class Employee {   //finalize() gets executed before an object is garbage collected
    ~~p s v m ()~~ @Override
    protected void finalize() {
        S.O.P("Eating Biryani");
    }
    p s v m () {
        Employee emp=new Employee();
        emp=null;
        System.gc();
    }
}
```

op :-
Eating Biryani

Q) WAJP to demonstrate java been class for a single variable where in class is Student and the attribute is marks.

Sol
```
public class Student {
    private int marks;
    ~~Student (int marks){~~
        ~~this.marks = marks;~~
    ~~}~~
    public void setMarks(int marks){
        this.marks = marks;
    }
    public int getMarks(){
        return marks;
    }
}
class Solution{
    p s v m ( ){
        Student s = new Student();
        s.setMarks(45);
    s.o.p(s.getMarks());
    }
}
```

Q) WAJP demonstrate constructor chaining in different classes

Sol
```
class Person {
    int id;
    Person (int id){
        this.id = id;
    }
}
```

```
class Student extends Person{
    int marks;
    Student (int id, int marks){
        super(id);
        this.marks = marks;
    }
}
class Solution{
    psvm(){
        Student s = new Student (101, 95);
        S.O.P("Id: "+s.id+" \n Marks: "+s.marks);
    }
}
```

```
class Person extends Object
?
    Pri ??
    {
    }
}
```

```
psvm(){
    new Pri();
}
```

CC

Q) WAJP to compare the name of 2 student objects after overriding equals().

Sol

```
class Student {
    @Override
    public boolean equals(){
        return
    }
    String name;
    Student (String name){
        this.name = name;
    }

    @Override
    public boolean equals(Object obj){
        Student s = (Student)obj;
        return this.name.equals(s.name);
    }
    psvm(){
        Student s1 = new Student ("Tom");
        Student s2 = new Student ("Tony");
        S.O.P (s1.equals(s2));
    }
```

29/08/2022

Anonymous Inner Class:-

* Anonymous inner class is a class without any explicit name.
* The name of the anonymous inner class is implicitly decl by the compiler.
* Anonymous inner class is generally used with abstract or interface.

Syntax:- ClassName reference = new ClassName(){

};

Traditional way of overriding an abstract method

```
abstract class Person{
    abstract void eat();
}
class Tom extends Person{
    @Override
    void eat(){
        S.O.P("Eating poori");
    }
    public static void main(String[] args){
        Tom t = new Tom();
        t.eat();
    }
}
```

O/P:-
Eating poori

Ex:-
```
package com;
abstract class Employee{
    abstract void work();
}
package com;
class Solution{
    p s v m () {
        Employee emp = new Employee(){
         @Override
         void work(){
             S.O.P("Working");
         }
        };
        emp.work();
}
```

O/P:-
Working

Ex:-
```
package com;
abstract class Car{
   abstract void start();
}
public class Test{
    p s v m ( ) {
      Car c = new Car(){
       @Override
       void start(){
           S.O.P("Car Started");
       }
       c.start();

       newCar ( ){
```

O/P:-
Car Started
Car Started

```
    @Override
    void start(){
        S.O.P("Car Started");
    }
}.start();
}
```

Ex:- Anonymous Inner class WRT interface

```
interface Flipkart{
    void purchase();
}
class Runner {
    p s v m(){
        Flipkart fk = new Flipkart(){
            @Override
            public void purchase(){
                S.O.P("Purchasing Mobile");
            }
        };
        fk.purchase();

        new Flipkart(){
            @Override
            public void purchase(){
                S.O.P("Purchasing Mobile");
            }
        }.purchase();
    }
}
```

O/P:-
Purchasing Mobile
Purchasing Mobile

Question

1) WAJP by following below scenarios

* Create a class called as Student
* Declare 2 attributes called as name & marks
* Initialize those attributes with the help of a constructor
* Override toString() to return the name & marks in the below format "Tom has scored 45 marks".

```
class Student {
    String name;
    int marks;
    Student (String name, int marks){
        this.name = name;
        this.marks = marks;
    }
    @Override
    public String toString (){
        return name + " has scored " + marks + " marks";
    }
}
import java.util.*;
class Test{
    p s v m ( ){
        Student s1 = new Student ("A", 55);
        Student s2 = new Student ("B", 26);
        Student s3 = new Student ("C", 89);
        Student s4 = new Student ("D", 45);
        LinkedHashMap<Student, Integer> map = new LinkedHashMap<Student, Integer>
        map.put(s1, 55);
```

```
map.put(s2,26);
map.put(s3,21);
map.put(s4,27);
Set<Student> keys = map.keySet();
//Printing keys and Value (Student Name, Marks & Age)
for(Student key : keys){
    S.O.P(key+" & age is "+map.get(key));
}
S.O.P("----------");
for(Student key : keys){
    ~~if(map.get(key)~~
    if(map.get(key) <25){
        S.O.P("Name:"+key.name);
    }
}
S.O.P("----------");
for(Student key:keys){
    if(key.marks>50){
        S.O.P("Name:"+key.name);
    }
}
}
}
```

O/P:-

```
A has scored 56 marks & age is 22
B has scored 26 marks & age is 26
C has scored 89 marks & age is 21
D has scored 45 marks & age is 27
----------
Name:
```

16) WAJP to store Hotel objects as the key and delivery time as the value within LinkedHashMap.

* Hotel objects will have 2 attributes (name & ratings)

Task 1:- Store 5 entries within the map & traverse it

Task 2:- Print the names of the hotel whose ratings are above 4.0

Task 3:- Print the names of Hotel who can deliver food within 30 minutes

Task 4:- Print the names of the Hotel whose ratings are above 4.0 & delivery time is lesser than 30 minutes.

```
class Hotel{
    String name;
    double ratings;
    ~~int deliveryTime;~~
    Hotel(String name, double ratings ~~int deliveryTime~~){
        this.name = name;
        this.ratings = ratings;
        ~~this.deliveryTime = deliveryTime;~~
    }
    @Override
    public String toString(){
        return "Name: "+name+ "\n Ratings: "+ratings;
    }
}
class Solution{
    p s v m ( ){
        Hotel h1= new Hotel("Taj",4.8);
        Hotel h2= new Hotel("ITC", 4.7);
        Hotel h3= new Hotel("Behrouz", 4.1);
        Hotel h4 = new Hotel("Empire", 3.7);
        Hotel h5= new Hotel("Savi Sagar", 3.5);
```

```
LinkedHashMap<Hotel, Integer> map = new LinkedHashMap<Hotel, Integer>();
map.put(h1, 25);
map.put(h2, 30);
map.put(h3, 35);
map.put(h4, 27);
map.put(h5, 32);
Set<Hotel> keys = map.keySet();
for(Hotel key : keys){
    S.O.P(key+" & deliverytime is "+map.get(key));
}
S.O.P("--------");
for(Hotel key : keys){
    if(key.ratings > 4.0){
        S.O.P("Name:"+key.name);
    }
}S.O.P("--------");
for(Hotel key : keys){
    if(map.get(key) < 30){
        S.O.P("Name:"+key.name);
    }
}S.O.P("--------");
for(Hotel key : keys){
    if(key.ratings > 4.0 && map.get(key) < 30){
        S.O.P("Name:"+key.name);
    }
}
}
}
```

Q) WAJP by following the below scenarios.

* Create a public non-abstract class called as Student
* Declare a private attribute called as age.
* Initialize the age with the help of a constructor
* Declare a public getter() for the attribute age.
* Override toString() to return the age.

Java Grooming Questions:-

1) WAJP to demonstrate static blocks & Non-static blocks.
2) WAJP to demonstrate java bean class.
3) WAJP to demonstrate Compile Time Polymorphism.
4) WAJP to demonstrate RunTime Polymorphism.
5) WAJP to demonstrate Abstraction.
6) WAJP to compare the attributes of 2 objects by overriding equals().
7) WAJP to demonstrate Single Ton class.
8) WAJP to demonstrate Custom Exception.
9) WAJP to sort objects using Comparable interface.
10) WAJP to sort objects using Comparator interface.

Q) WAJP to check if a number is even/odd.

```
class EvenOrOdd{
    p s v m ()
    int n=2;
    if(n%2==0)
        S.O.P("Even");
    else
        S.O.P("Odd");
    }
}
```

2Q) WAJP by following the below scenarios

a) Create a public class called as Jspiders.

b) private data member called as branch.

c) Develop respective public getter() & setter().

```
public class Jspiders{
    private String branch;
    ~~Jspiders(String branch){~~
        ~~this.branch = branch;~~
    ~~}~~
    public void setBranch(String branch){
        ~~return "Branch:"+branch;~~ this.branch = branch;
    }
    public String getBranch(){
        return branch;
    }
    class solution{
    P S V M ( ) {
        Jspiders j = new Jspiders();
        j.setBranch("Basavanagudi");
        j.getBranch();
    }
}
```

MCQ Questions:-

1) Can static methods can be overloaded? [True]

2) Can we overload non-static methods? [True]

3) Can we override non-static method? [True]

4) Can static methods be overridden? [False]

Static methods belongs to the class, therefore it cannot be Overridden.

5) Abstract methods can be overridden? [True]
6) Abstract methods can be static? [False]
7) Abstract methods can be private? [False]
8) Abstract methods can be final? [False]

Note:- Abstract methods can not be private, static and final because abstract methods need to be overridden.

9) Java supports nested class? [True]
10) Java supports nested methods? [False]
11) Try and catch block can be present in finally block? [True]
12) Java supports nested try and catch block? [True]
13) We can have executable lines between try & catch block [False].
14) We can have comments b/w try & catch block [True].

7/9/2022

1) Variables & types of variables
2) Datatypes & types of datatypes
3) Working of a java program (WORA) architecture.
4) this keyword 5) default values
6) Diff b/w static & non-static.

12/9/22

1) WAJP by following the below scenarios
a) Create a class called as Employee
b) Declare 2 attributes called as name & salary
c) Initialize the attributes with the help of a constructor
d) Override toString() to return the name & salary.

```
Class Employee {
    String name;
    double salary;
    Employee(String name, double salary){
        this.name = name;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Name: " + name + " Salary: " + salary;
    }
}
Class Solution {
    static void findHighestSalary(Employee e1, Employee e2, Employee e3)
    {
        if((e1.salary > e2.salary) && (e1.salary > e3.salary)){
            S.O.P(e1.name + "has the highest salary");
        }
        else if((e2.salary > e1.salary) && (e2.salary > e3.salary)){
            S.O.P(e2.name + "has the highest salary");
        }
        else {
            S.O.P(e3.name + "has the highest salary");
        }
    }
    p s v m ( ){
        Employee e1 = new Employee("Tom", 12.5);
        Employee e2 = new Employee("Jack", 12.2);
        Employee e3 = new Employee("Alex", 12.9);
```

```
S.O.P(e1);
S.O.P(e2);
S.O.P(e3);
S.O.P("--------");
findHighestSalary(e1,e2,e3);
}
}
```

2) WAJP by following the below scenarios

a) Create a class called as Hotel

b) Declare 2 attributes called as name & ratings

c) initialize the attributes with the help of a constructor.

d) Create another class called as Solution

e) Declare a non-static method called as findLowestRatings(). which accepts 3 hotel objects & find the name of the hotel with the lowest ratings

```
class Hotel{
    String name;
    double ratings;
    Hotel(String name, double ratings){
        this.name=name;
        this.ratings=ratings;
    }
}
class Solution{
    void findLowestRatings(Hotel h1, Hotel h2, Hotel h3)
    {
        if(h1.ratings < h2.ratings && h1.ratings < h3.ratings){
            S.O.P(h1.name+"has lowest ratings");
        }
```

```
else if(h2.ratings<h1.ratings && h2.ratings<h3.ratings)
    S.O.P(h2.name+"has lowest ratings");
}
else {
    S.O.P(h3.name+"has lowest ratings");
}
}
psvm( ){
    Hotel h1=new Hotel("Taj",4.7);
    Hotel h2=new Hotel("Empire",3.8);
    Hotel h3=new Hotel("ITC",4.5);
    findLowestRatings(h1,h2,h3);
}
}
```

3) WAP to store student & employee objects into Arraylist & traverse it using enhanced for loop.

* Student will have name & marks
* Employee will have name & salary
* Arraylist should not allow objects other than student & employee

```
class Student extends Person {
    String name;
    int marks;
    Student(String name, int marks){
        this.name;
        this.marks;
    }
```

```
class Employee Extends Person {
    String name;
    double salary;
    Employee(String name, double salary){
        this.name = name;
        this.salary = salary;
    }
}
class Person {
}
class MainClass {
    psvm() {
        Student s1 = new Student("Tom", 56);
        Student s2 = new Student("Jerry", 89);
        Employee e1 = new Employee("Alex", 1.2);
        Employee e2 = new Employee("Basil", 1.5);
        ArrayList<Person> al = new ArrayList<Person>();
        al.add(s1);
        al.add(s2);
        al.add(e1);
        al.add(e2);
        for(Person p : al){
            if(p instanceof Student){
                Student s = (Student) p;
                S.O.P("Name:"+s.name+" Marks:"+s.marks);
            }
            else if(p instanceof Employee){
                Employee e = (Employee) p;
                S.O.P("Name:"+e.name+" Salary:"+e.salary);
            }
        }
    }
}
```

Q) Explain System.out.println()?

Ans * S.O.P() is a pre-defined example for HAS-A Relationship (Association).

* System is a pre-defined class present in java.lang package
* System class has a static reference variable "out" of type PrintStream which is an instance of PrintStream class
* PrintStream is a pre-defined class present in java.io package.
* println() is a pre-defined non-static method present in PrintStream class.

Note:-

* Println() is a pre-defined example for method overloading.
* Both print() & println() of PrintStream class ~~is an~~ are both example for method overloading.

```
class System
{
  static PrintStream out = new PrintStream();
      System.out.println();
}
```

```
class PrintStream
{
  void println()
  {
  }
  void print()
  {
  }
}
```

Overloading examples

Note:-

* toString(), hashCode() & equals() are pre-defined examples for methodoverriding.
* The above mentioned 3 methods are automatically overrideden in String class.

Few more examples for method overloading:-

* wait() present in object class
* add() present in Collection interface

Few more examples for method overriding:-

* All Wrapper classes & String class implements Comparable interface and has overridden compareTo().

* All the classes in java is a pre-defined ~~method~~ example for IS-A Relationship (Inheritance).

Ex:

```
interface Vehicle {
    void start();
}
class Car implements Vehicle {
    @Override
    public void start() {
        S.O.P("Car Started");
    }
}
class Bus implements Vehicle {
    @Override
    public void start() {
        S.O.P("Bus Started");
    }
}
class Bike implements Vehicle {
    @Override
    public void start() {
```

```
        S.O.P("Bike Started");
    }
}
class Person {
    Vehicle obj = new Car();  //UPCASTING -> we can store new Bus() {
                                                  new Bike()
                                                  also
}
class Solution {
    p s v m () {
        Person p = new Person();
        p.obj.start();
    }
}
```

O/P :-
Car Started

or

```
class Person {
    Vehicle vehicle;  //Car or bike or bus
    Person(Vehicle vehicle) {
        this.vehicle = vehicle;
    }
}
class Test {
    p s v m () {
        Bike bike = new Bike();
        Bus bus = new Bus();
        Car car = new Car();
        Person p = new Person(bus);  // Person p = new Person(new Bus());
        p.vehicle.start();  //Bus Started
    }
}
```

O/P :-
Bus Started