INDEX

SR. NO.	DATE	
		Interface.
22.		Package (Pre final mock)
23		Access specifiers.
24.		Encapsulation.
25.		Collection
26		Object class.
27.		String class
28.		Exception handling.
29.		Threads.
30.		File handling (final mock)
31.		

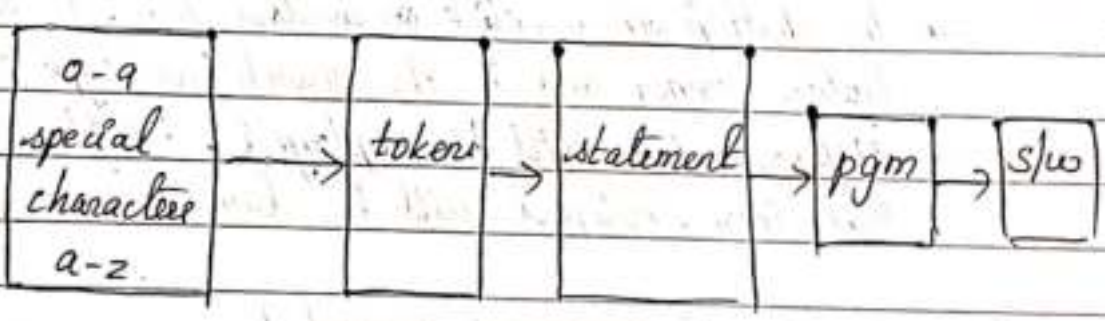
Language –

Language is a medium to communicate between two entities.

Programming Language –

It is a medium to communicate between a human and a machine. We have various programming languages like C, C++, Java, Python and C#.

How a programming language is built?



Softwares Required :-

1. Editors

* Notepad.
* Editplus.

2. Java Software.

Jdk (Java Development Kit)

Jdk 1.0	Jdk 1.6	Jdk 1.12
1.1	1.7	1.13
1.2	1.8	
1.4	1.9	
1.5/5.0	1.10	
	1.11	

3. IDE (Integrated Development Environment)

* Eclipse → SE (Standard Edition)
  → EE (Enterprise Edition)
  → ME (Micro Edition)

* Netbeans

4. Execution Area
   cmd / command prompt.

Q. How To launch Notepad?

→ On the system.

→ In desktop screen, click on windows icon, in the left bottom corner and in the search bar type 'Notepad'

→ Notepad icon will be displayed. Double click on that icon, notepad will be launched.

Q. How to launch command prompt?

→ On the system.

→ In duktop screen click on windows icon. In the left bottom corner search bar type 'cmd' or 'command prompt'.

→ Command prompt will be displayed, double click on that icon, command prompt will be launched.

Commands used in Java :-

1. cls - clear screen
2. cd - change directory.
3. cd.. - change from current directory to previous directory.
4. mkdir - make directories
5. Java c - Java complier.
6. Java - Java Interpreter.

TOKENS :

It is a smallest unit of program.

① Identifiers
② Literals
③ Comments
④ keywords
⑤ Operators
⑥ Seperators.

① Identifiers :

Identifiers are the names given for any java program.

② Literals :

Literals are the values which are used in java programming language.
In Literals we have

i) Numeric Literals
ii) Character Literals
iii) String Literals
iv) Boolean Literals

i) Numeric Literals -

(a) Integer : Any number without decimal value is called as Integer literals.

eg- 9, 10, 25, 36.

(b) Decimal : Any number with decimal value is called as Decimal literals.

eg- 5.56, 76.36, 89.48.

ii) Character Literals -

Anything which is enclosed with single codes is called as Character Literals.

We cannot have more than one character in character literal.

eg- 'A', 'S', 'T', '9', ' ', 'abc'

iii) String Literals -

Anything which is enclosed with double codes is called as String Literals.

eg- "Hello", "abc123$", "", "A"

iv) Boolean Literals -

In java we have only 2 values and that is true or false.

③ Comments :

Comments are used to provide the additional information or to skip the particular line of code

In comments we have

① Single line comment

// -------

② Block comment

/* ---------- */

④ Keywords :

Keywords are the predefined words. It has its own meaning. In java we have 50 keywords as follows :

1. Abstract	12. default	22. goto*	32. package
2. assert***	13. do	23. if	33. private
3. boolean	14. double	24. implements	34. protected
4. break	15. else	25. import	35. public
5. byte	16. enum****	26. instance of	36. return
6. case	17. extends	27. int	37. short
7. catch	18. final	28. interface	38. static
8. char	19. finally	29. long	39. strictfp**
9. class	20. float	30. native	40. super
10. const*	21. for	31. new	
11. continue.			
41. switch	49. while.		
42. synchronized	50. throws.		
43. this			
44. throw			
45. transient			
46. try			
47. void			
48. volatile.			

* → not used

** → added in 1.2

*** → added in 1.4

**** → added in 5.0

(3) Operators :

Operators are the symbols which is used to perform some operation.

example: 3 + 10

OPERATOR TYPE	CATEGORY	PRECEDENCE
unary	post fix	expr++ expr--
	pre fix	++expr --expr +expr
		-expr ~!
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	Comparison	< > <= >= Instance of
	equality	== !=
Bitwise	bitwise AND	&
	bitwise inclusive OR	|
	bitwise exclusive OR	^
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /=
		%= &= ^= |= <<=
		>>= >>>=

⑥ Seperators

Seperators are used to seperate the given Java code

a. braces { }
b. brackets [ ]
c. parantheis ( )
d. semicolon ;
e. comma ,

Q How do download Java software?

→ On the system and launch the browser.
→ In the google search box type download Jdk 1.8
→ Click on search button
→ In the search results clicks on first link. It will take to the oracle download page.
→ Scroll and click on accept licesense agreement.
→ If your laptop is 32 bit then download win x86. If laptop is 64bit download win x 64.

Q. How to find laptop's 32 bit or 64 bit?

Right click on this PC and click on properties

HISTORY OF JAVA

James Gosling is the founder of Java in the year 1991 and the software was named as Green Talk. The team which develop this software was named as Green Team.

Later they renamed as Oak. Oak is the symbol of strength and it is a National Tree of Germany.

At the same time there was already an existing company called Oak Technologies which became a legal issue and changed the software name from Oak to Java in the year 1995.

James Gosling and his team went to an Island for a coffee and the coffee shop name was Java and hence they kept the software name as Java and the logo as a coffee mug.

JAVA ARCHITECTURE

OR

HOW A JAVA PGM WILL GET EXECUTED INTERNALLY

As a TE we will write the pgm in notepad or editplus, it is also called as source code which is in human readable format.

Once after writing the pgm, save the pgm with .java extension in the below mentioned path.

c:\pgm files\java\Jdk1.8.0_221\bin

To convert this human readable format into machine readable format we have to perform 2 operations in the command prompt. That is

i) Java Compiler (Javac)
ii) Java Interpreter (Java)

The .java file will be given as an input for the compiler which checks for

i. syntax
ii. rules
iii. translates from .java file to .class file

If we voilate any syntax or rules we will get compile time error (CTE).

Once the g .class file is generated it is an intermediate code which is in byte code format where it cannot be understood neither by human nor by the machine.

The .class file will be given as an input for the interpreter which will give the binary output and intern given to the OS to generate the output.

i Interpreter will read line by line,
ii execute (JVM)
iii translates from .class file to binary format.

During execution if we find any abnormal statements like 1/0, we get arithmetic exception. that is run time error.

JIT (Just in Time):

It is whole responsible to convert from .class file to binary format.

JVM (Java Virtual Machine):

It is a virtual machine which doesn't exists physically and it is whole responsible to execute the Java pgm.

JRE (Java Runtime Environment):

It is an environmental setup provided to run the Java pgm.

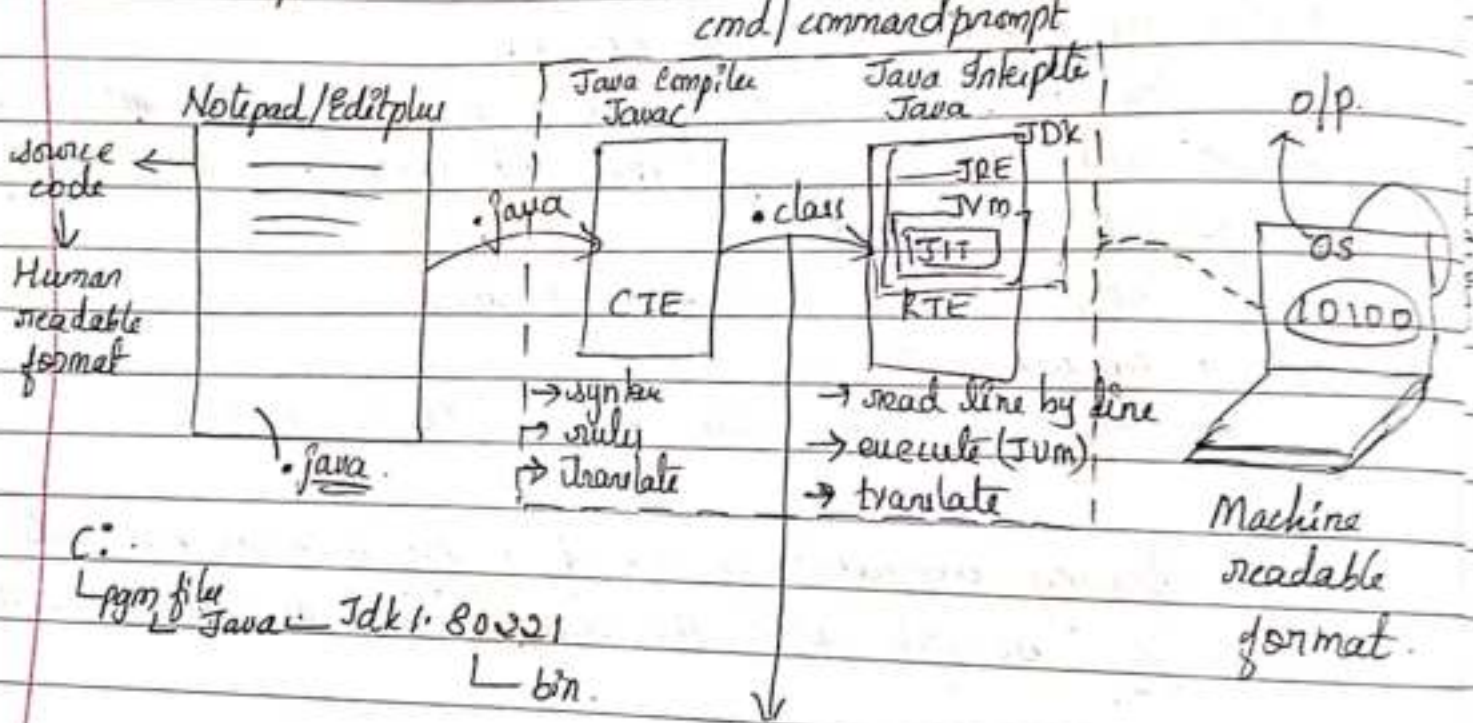
Q. Why Java is platform independent ?

→ JRE is installed in every electronic device, hence the .class file can be executed on any electronic device, hence java is platform independent.

JDK (Java Development Kit):

It is kit which consists of all the library files and utilities to develop a java s/w.

Always the java pgm will execute from left to right, top to bottom.

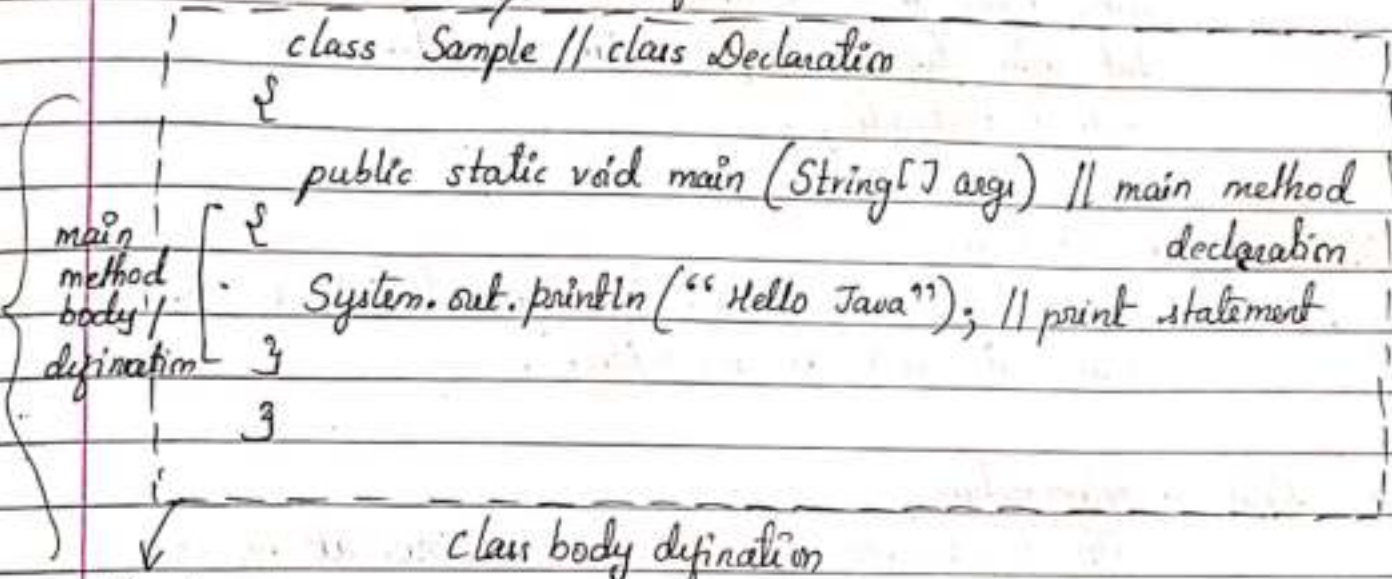


JAVA TYPES

In java we can write the programs in 4 types

1. class
2. enum
3. Annotation
4. Interface

1. CLASS :



Program :-

```
class Sample
{
public static
```

command for Interpreter
syntax: java file_name
Ex: java Sample

Command fo compilation
syntax: javac file_name.java
Ex: javac Sample.java

In command prompt

```
C:\Users\admin>d:
D:\>cd javamavabatch
D:\javamavabatch>cd intro
D:\javamavabatch\intro>javac Sample.java
D:\javamavabatch\intro>java Sample
Hello java
```

Interview Questions :-

Q. What is method overloading?

Developing multiple methods with a same name but variation in arguments list is called as 'method overloading'

Q. What is method overriding?

Developing a method in a sub class with the same name and signature as in the super class but with different implementation is called as 'method overriding'.

Q. What is inheritance?

Inheriting the properties from one class to another class is called as "inheritance".

Q. What is polymorphism?

An object showing different behaviour at different stages of its life cycle is called as polymorphism.

Q. What is encapsulation?

Declare the data ~~number~~ member as private and restrict the direct access outside the class and provide the indirect access through public services called getters and setters is called as encapsulation.

Q. What is collection?

Collection is an unified architecture which consists of classes and interfaces.

What is Abstraction?

Hiding the complexity of the system and exposing only the required functionality is called as abstraction.

Resume points :-

→ Good knowledge in Method Overloading

→ Very good understanding on method overriding.

How To Install Java Software?

1. Double click on downloaded exe file.
2. Click next next next finish

How to set the path:

→ Once after java s/w is installed, go to the below path

C:\Program Files\Java\jdk1.8.0_211\bin right click & Copy the path, right click on this PC – click on properties – advanced system setting – environmental variables, under

→ Under system variable select path, click on edit, click on new, paste it and click ok 3 times.

How to check java s/w is installed in laptop or not

→ Close the cmd prompt if it is open

→ launch the cmd prompt, Type java and javac. If you get the lengthy message then java s/w is installed correctly.

Program:

```
class Demo
{
public static void main(String[] args)
{
system.out.println(20); → op 20
system.out.println(8.68);
system.out.println("Hi");
system.out.println('S');
system.out.println(True);
}
}
```

② public class Sample.

```
{
public static void main (String[] args)
{
system.out.println (20);
system.out.println (20+20);
system.out.println ("it is a number" + 20 + 20);
system.out.println (20+20+ " it is a number");
system.out.println ("it is a number" + 20.34);
system.out.println ("it is a number" + 1/2);
system.out.println ("it is a number" + 1/2.0);
system.out.println ("it is a number" + 1.0/2.0);
}
}
```

Output:-

20
40
it is a number 40
40it is a number
it is a number 20.34
It is a number 0
It is a number 0.5
it is a number 0.5

→ + (Plus) operator will work as addition operator as well as concatenation operator.

→ If we use the plus operator after a string it will act as a concatenation operator

→ If we use the plus operator before the string along with a number, it will act as a concatenation operator.

→ If we are performing the addition operation between 2 numbers before a string, the plus operator between those 2 numbers will act as a addition operator and compulsory we have to use plus operator betn the o/p and the string.

print

① Write a program to printed a character, integer number, decimal, boolean and string.

→ class Sample
{
public static void main (String[] args)
{
system.out.println ("Hello java");
system.out.println (10);
system.out.println (10.50);
system.out.println ('A');
system.out.println (true)
}
}

output:

Hello java
10
10.50
A
true

→ class Sample
{
public static void main (String[] args)
{
system.out.println ("the value is" + (1/2.0));
system.out.println (20);
system.out.println (20+20);
system.out.println ("the value is" + 20);
system.out.println ("the value is" + 20+20);
system.out.println ("the value is" + (20+20));
system.out.println (20 + "is the value");
system.out.println (20+20+ "the value");

System.out.println("the value is" + (1/2));
}
}

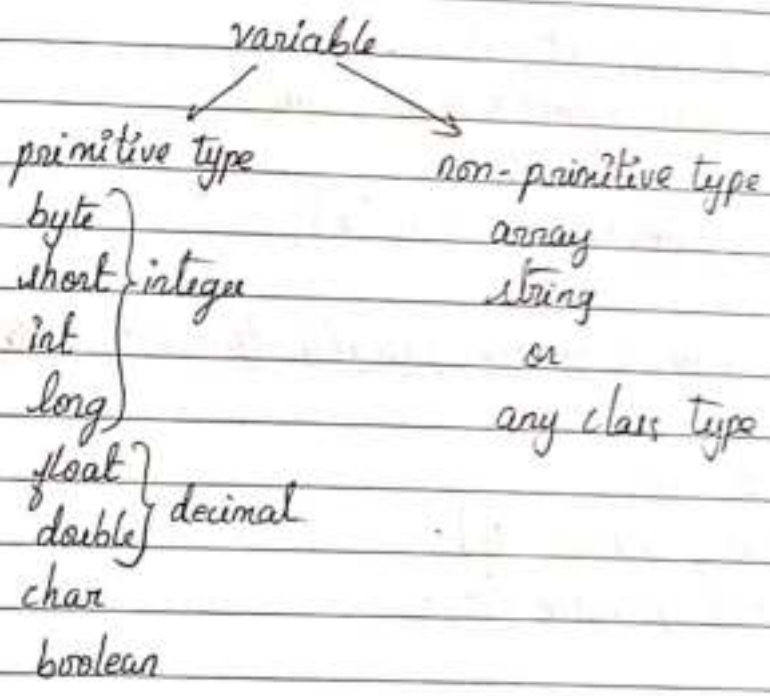
output:

20 the value is 0.5
20
40
the value is 2020
the value is 40
20 is the value
40 is the value.
the value is 0

VARIABLES

Variable is a named memory location which can store some value or data and it can change n no of times during execution.

Variables are of 2 types
i) Primitive type.
ii) Non Primitive type
or
Reference type
or
Class type.



Variable Declaration:
syntax - datatype Variable_name;
ex - int a;
int _ab;
int $xyz;

Variable Initialization:

syntax — Variable_name = value;

ex — a = 10;

Variable Utilization:

syn'

system.out.println(a);

Variable Declaration and Initialization in a Single line:

syntax: datatype Variable_name = Value;

int x = 40.

system.out.println(x);

Variable Re-Initialization:

syntax: Variable_name = new value;

x = 80;

system.out.println(x);

Copying the value from one variable to another variable:

int y = 70;

int z = y;

system.out.println(y);

system.out.println(z);

class Sample
{
public static void main (String[] args)
{
System.out.println (20.5e-2);
System.out.println (20.5e+2);
System.out.println (20.56);
}
}

output:
20.56
20.56
20.56

Variable can be classified into:-
i) Local Variable
ii) Global Variable

i) Local Variable :-)
Any variable which is declared either the method is called as "Local Variable".

The scope of the local variable is from the opening of the method till the end of the method

It cannot be changed into static and non static.

It will not have default values.

Local variable should be initialized upon otherwise else we get compile time error.

example: class Demo
{
public static void main (String [] args)
{
int a;
a = 10;
system.out.println(a);
}
}

ii) Global Variable :→

Any Variable which is declared outside the method, inside the class is called as 'Global Variable'

The scope of the global variable is from the beginning of the class till the end of the class

It can be classified into static and non static

It will have default values.

Once the global variable is declared immediately in the next line we cannot initialize or re-initialize, else we get compile time error.

example: class Sample
{
int empid = 123
public static void maid (String [] args)
{
empid = 123;
system.out.println(empid);
}
}

Primitive Data type	Default value
byte	0
short	0
int	0
long	0
float	0.0f
double	0.0d
char	'/u0000'
boolean	false
string	null

What is the difference between byte, short, int, long, float, double?

Example: (how to count for byte)

byte — 1 byte → 8 bits

-2^{n-1} to $2^{n-1}-1$

-2^{8-1} to $2^{8-1}-1$

-2^{7} to $2^{7}-1$

-128 to $128-1$

$\boxed{-128 \text{ to } 127}$

Type	Size (in bit)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-8$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true, false

→ Kirti joined Qspiders and he started with manual testing and got the mock rating as 1. Along with Kirti, Kavya also joined and started doing Java and got mock rating 1.
Write a program to print institute name, subjects name, student names and the mock rating.

```
class Sample
{
public static void main (String[] args)
{
    String instti = "Qspiders";
    String std1 = "Kirti";
    String std2 = "Kavya";
    int msld1 = 1;
    int mstd2 = 1;
    String substd1 = "Manual Testing";
    String substd2 = "Java".
system.out.println (std1 + "subject is" + substd1 + "
        & mock is" + mstd1);
system.out.println (std2 + "subject is" + substd2 "
        + & mock is" + mstd2);
}
}
```

In a class we have 3 members

i) Variables or Data members –

It is used to store data or value

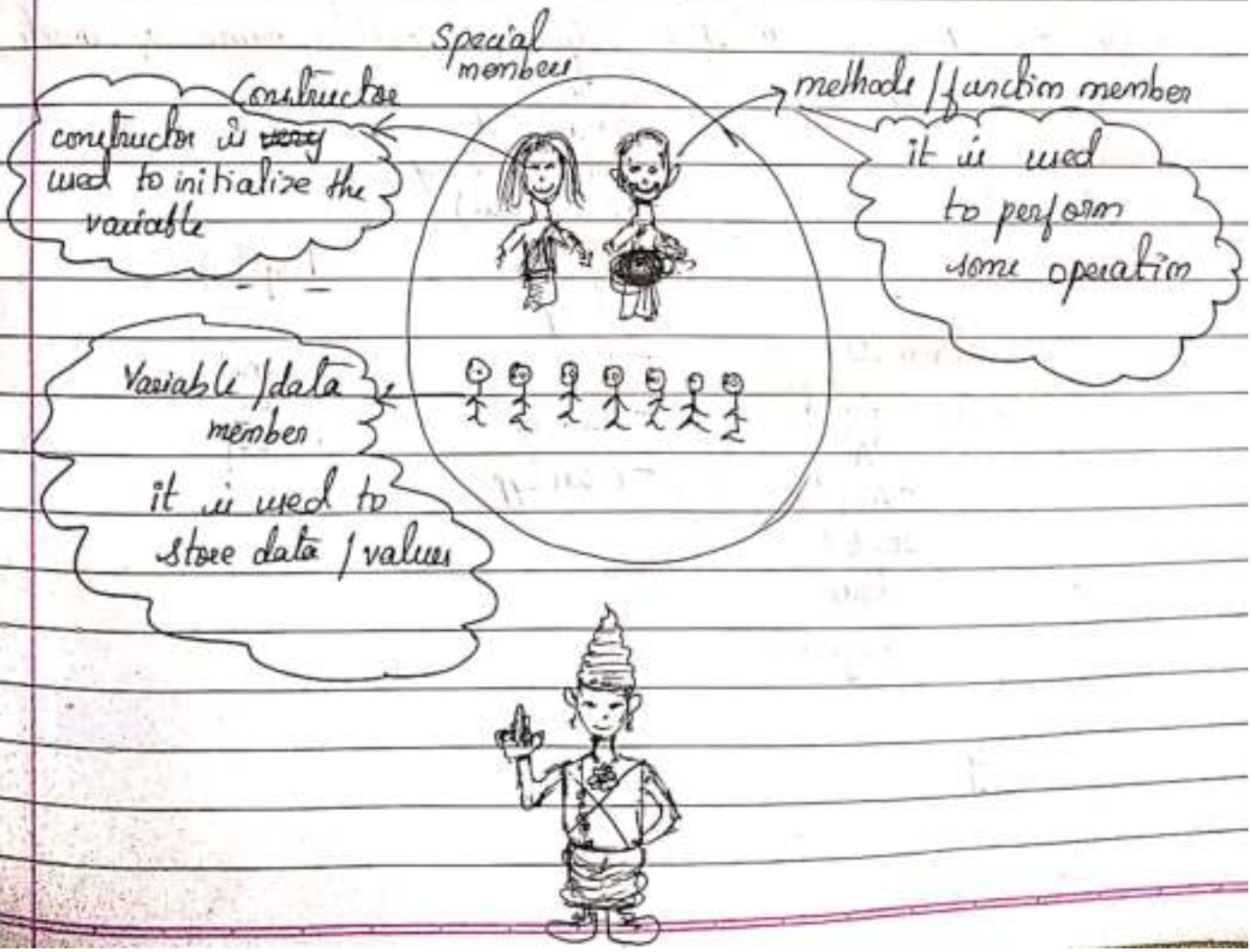
ii) Method or Function member –

It is used to perform some operation

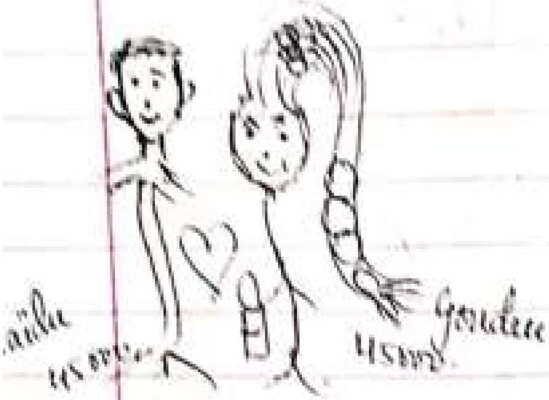
iii) Constructor –

It is used to initialize the variable or data members.

```
class Demo
{
  Variable / data member
  Method / Fn member
  Constructor
}
```



METHOD



Method is a block of statement which will get executed whenever it is called or invoked

syntax:

public / protected / default/package / private — Access specifier

static / non static — Modifier

return-type → void (doesn't return any value) / int / double / char / string / float / boolean / class type

method_name → identifier

(arguments) → optional

{

return 10;

return 20.5d

" 'A'

" "Abc"

" 20.6f

" true

" object

}

dispText

camel case

OCJp

arguments syntax :-

(datatype Variable_name)

Ex : (int a)

(int x, double y)

```
class Demo
{
static void add()
{
    int a = 10;
    int b = 20;
    int c = a+b;
system.out.println(c);
}
 public static void main (String[] args)
  {
system.out.println(---- Main Starts ---);
  add();
system.out.println(---- Main Ends ----);
 }
}
```

→ Write a program to multiply 2 numbers

```
class Multiply
{
  static void mul()
  {
    int x = 4;
    int y = 2;
    int z = x * y;
  system.out.println(z);
}
    public static void main(String[] args)
  {
      system.out.println("--- Main begins ---");
        mul();
        system.out.println("---- Bye main ---");
    }
  }
```

→ Write a program to calculate area of a circle.

```
class Circle
{
  static void area()
  {
    int r = 5;
final double pi = 3.142;
  double result = pi * r * r;
  system.out.println(result);
}
  {
    public static void main(String[] args)
{
```

data	operation
int r = 5	multiplication
final double pi = 3.142;	pi * r * r

```
System.out.println("--- Main Start ---");
    area();
System.out.println("---Main End ---");
}
}
```

Methods with Parameters:

Whenever we want to provide input for the method then we should go for "method with parameters".

① Write a program to add 2 number with method with parameters.

→

```
class Demo
{
static void add(int a, int b) // parameters
{
    int c = a+b;
    system.out.println(c);
}
public static void main(String[] args)
{
System.out.println("---- Main Start ----");
add(20,30) // arguments
System.out.println("---- Main Ends ----");
}
}
```

② Write a program to multiple 3 integer numbers method with parameters

→

```
class Multi(
{
static void multi(int a, int b, int c)
{
    int d = a*b*c;
    System.out.println(d);
}
public static void main(String[] args)
{
```

```
System.out.println("----- Main Start -----");
multi(50, 60, 30);
System.out.println("----- Main Ends -----");
}
}
```

data | operation

if we want to add some value or data we go for variables

if we want to perform some specific then methods.

```
class Circle
{
static void area(int r)
{
final double pi = 3.142;
double result = pi * r * r;
System.out.println("area is " + result);
}
public static void main(String[] args)
{
System.out.println("----- Main Start ---");
area(6); area(10);
System.out.println("----- Main Ends ----");
}
}
```

METHOD WITH RETURN TYPE :

Whenever we want the result of an operation in the method to another method then we should go for method with return type.

Real Time Example :-

Rajesh joined IBM with the salary 10,000 Rs and he worked for 1 year, since he was the best employee, he got a bonus of 5000 Rs.

The account team will not make the decision on the bonus amount. The manager is going to take a call whose is the best employee and how much is the bonus amount.

Write a pgm to get the yearly salary from the account department and add the bonus, display total annual salary received by Rajesh.

```
→ class Demo
{
  static int disp()
           (bontype)
  {

     return 10;   (pizza)
  }

  public static void main(String[] args)
  {
     int x = disp();
     sop(x);
  }
}
```

```
class IBM
{
static int accounts()
{
    int m_sal = 10000;
    int no_months = 12;
    int result = m_sal * no_months;
                 10000 * 12

    return result;
}

public static void main(String[] args)
  {
   int y = accounts();
   Int bonus = 5000;
   Int total = y + bonus;
               120000 + 5000

   System.out.println(total);
}
}
```

data	operation
int m_sal = 10000	multiplication
int no_mon = 12;	+
int bonus = 5000	addition.

→ Write a pgm to add 2 no's and return the value to the main method and print it.

```
class Add
{
    static int addition ( )
    {
        int a = 10;
        int b = 10;
        int c = a+b;
        return c;
    }
    public static void main (String [] args)
    {
        int y = addition ( );
        System.out.println (y);
    }
}
```

hide this — only this is visible

This is visible only within the method. we cnt use 'c' anywhere bcz it is local variable

→ Write a pgm to calculate area of a circle with method with return type. πr^2

```
class Circle
{
    static double area ( )
    {
        int r = 8;
        final double pi = 3.142;
        double result = pi * r * r;
        return result;
    }
    public static void main (String [] args)
    {
        double z = area ( );
        System.out.println (z);
    }
}
```

CONDITION STATEMENTS

Conditional Statements are used to check logical conditions. We have <u>if condition</u>:

If condition will return true or false and the syntax is:

(1) <u>if Condition</u>

syntax:-

```
if (condition)
{
    ==========
}
```

only if it is true then only this condition get executed

example:-

```
if (5 > 3)
{
    System.out.println("Hi");
}
```

true

(2) Else Condition:-

```
if (condition)
{
    ==========
}
else
{
    ==========
}
```

true / false

example:-

```
if (5 > 3)
    SOP("5 is greater");
else
{
    SOP("5 is not greater");
}
```

③ Else if Condition

syntax

```
if (condition)
{
}
else if (condition)
{
}
else
{
}
```

example:

```
String expres = "Home page";
String actres = "Home page";
if (expres == actres)
{
SOP("Test case is True)
else
{
SOP("TC is false)
}
}
```

Condition

example.

④ Nested if else

syntax:

```
if (condition)
{
  if (condition)
  {
  }
  else
  {
  }
}
else
{
}
```

Home

LOOP :-

Whenever we want to perform any operation repeatedly for 'n' no. of times, then we shud go for loop.

FOR LOOP :-

Whenever we want to perform any operation with 'n' no. of iterations, then we should go for 'for loop'.

→ Write a pgm to print Hello Java 4 times.

```
class Demo3
{
public static void main (String[] args)
{
SOP ("Hello Java");
SOP ("Hello Java");
SOP ("Hello Java");
SOP ("Hello Java");
}
}
```

All this shud be written in an optimized way to reduce the no. of lines

If we have a range and if it has starting & ending then we go for 'for loop'

syntax :-

```
for(initialization; condition; increment/decrement)
{
==
}
```

example :

```
for (int i=1; i<=4; i++)
{
System.out.println ("Hello Java");
}
```

o/p :

Hello Java
Hello Java
Hello Java
Hello Java

```
class Sample1
{
public static void main (String [] args)
{
  int i=1;  ---> we can initialize here also
  for ( ; i<=4; )
  {
    System.out.println("Hello Java");
    i++; ---> we can increment here also.
  }
}
}
```

for loop is applicable for immediate next line.

```
for ex:- for (int i=1; i<=4; i++)
         {
           SOP (Hello) → only this will be print
         }SOP (Hii) → this will print only once.
```

braces shud be present for both Hello & Hi then only it will executed 4 times

```
ex:- for (int i=1; i<=4; i++)
     {
       SOP (Hello)
       SOP (Hii)
     }
```

→ class Sample1
{
public static void main (String[] args)
{
for (int i=1; i<=4; i++)
{
System.out.println ("Hello");
System.out.println ("Hii");
}
}
}.

Or

class Sample1
{
public static void main (String[] args)
{
for (int i=1; i<=4; i++)
SOP ("Hello");
SOP ("Hello");
}
}

SWITCH CASES

Switch case is used for pattern matching. It matches with the character sequence and executes the particular case.

Syntax:-

```
switch (charSeq)
{
case charseq : statement 1;
               break;
case charseq : statement 2;
               break;
case charseq : statement 3;
               break;
default : statement;
          break;
}
```

Example:

```
class Demo
{
public static void main (String [] args)
{
  char input = 'A';
  switch (input)
{
  case 'A' : SOP ("FCD");
             break;
  case 'B' : SOP ("FC");
             break;
  case 'C' : SOP ("SC");
             break;
```

```
case 'D' : SOP ("Go home");
           break;
default : SOP ("invalid input");
          break;
```

Suppose if there is no break mentioned after SOP ("FCD") then the o/p will be FCD & FC

- class Sample 9
{
public static void main (String[] args)
{
int input = 1
switch (input)
{
case 1 : SOP ("Chicken Biriyani");
break;
case 2 : SOP ("Mutton Biriyani");
break;
case 3 : SOP ("Egg Biriyani");
break;
case 4 : SOP ("Fish Fry");
break;
default : ("invalid input");
break;
}
}
}

→ Write a pgm to print from 1 to 10 using for loop

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

→ If we want to print i value then should put it in " " . .

→ Write a pgm to print from 20 to 30 using for loop

```
class Demo1
{
public static void main (String [] args)
{
for (int j=20; j<=30 ; j++)
{
System.out.println (j);
}
}
}
```

→ Write a pgm to print from 10 to 1

```
class Demo2
{
public static void main (String [] args)
{
   for (i = 10 ; i >= 1 ; i--)   (i=10; i>=1; i--)
   {
      System.out.println (i);
   }
}
}
```

10>=1 = 10
9>=1 = 9.....

→ Write a pgm to print Hello Java along with " "

```
class Demo3
{
public static void main (String [] args)
{
   System.out.println ("\"Hello Java\"");
}
}
```

→ Write a pgm to print Hello Java without using ;

```
class Demo4
{
public static void main (String [] args)
{
   for (i=1; i<=1; System.out.println ("Hello Java"))
   {
      i++;
   }
}
}
```

→ class Demo5
{
public static void main (String [] args)
{
if (true) ——→ if (5 > 2) internally (true)
{
System.out.println ("Hello");
}
}
}
O/P : Hello

→ class Demo6
{
static boolean add ()
{
return true;
}
public static void main (String [] args)
{
if (add())
{
SOP ("Hello");
}
}
}

O/P : Hello

replaced with true

```
class Demo
{
public static void main (String[] args)
{
int j=10;
System.out.println(j);
for(i=9; i>=9; i++)
{
System.out.println(i);
}
}
}
```

output: 10, 9

```
class Demo
{
public static void main (String[] arg)
{
int j=5;
SOP(j);
{
for(i=1; i<=4; i++)
SOP(i);
for(k=6; k<=10; k++)
SOP(k);
}
}
}
```

output: 5, 1, 2, 3, 4, 6, 7, 8, 9, 10

```
for(i=10; i>=7; i--)
SOP(i);
for(j=1; j<=6; j++)
SOP(j);
```

10 9 8 7 1 2 3 4 5 6

Scanned by CamScanner

STATIC

In the class members, variables and methods can be classified into static and non static. Constructor is always non static

```
class...
{
variable / data members   } static & non static
methods / function members }
constructor
    ↓
  always
  non-static
}
```

What is Static?

→ Any member of the class declared with static keyword is called as static member of the class.

→ Static is always associated with class.

→ Static is one copy

→ Whenever we want to access the static members from one class to another class then we have to use classname.variable name or class name.method name.

→ All the static members will be stored in static pool area

Example :-

```
class Tester
{
static int a = 10;
static void add ()
{
SOP ("Hello");
}
public static void main (String [] args)
{
add ();
SOP (a);
}
}
```

POST INCREMENT AND PRE INCREMENT

POST INCREMENT : (Use it and then increment)

i++ 6

int i = 1

i = i++ + i++ + i++ + i++ + i++ + i++ + i++

1 1 2 3 4 5 5 6

→ int i = 1

i++ + i + i++ + i++ + i + i++ + i++ + i++

1 2 2 3 4 4 5 6 (7)

= 27

→ int i = 1

i = i++ + i++ + i + i + i + i++ + i++ + i++

1 2 3 3 3 3 4 5

= 24

→ i = i++ + i + i++ + i++ + i + i++ + i + i++

1 2 2 3 4 4 5 5

= 26

PRE INCREMENT (Increment and then use it):

```
int i=1
i = ++i + i + ++i + ++i + i.
    2     2     3     4     4
  = 15
```

```
i = ++i + ++i + ++i + ++i + i + ++i + i + ++i
    2     3     4     5     5     6     6     7
  = 38
```

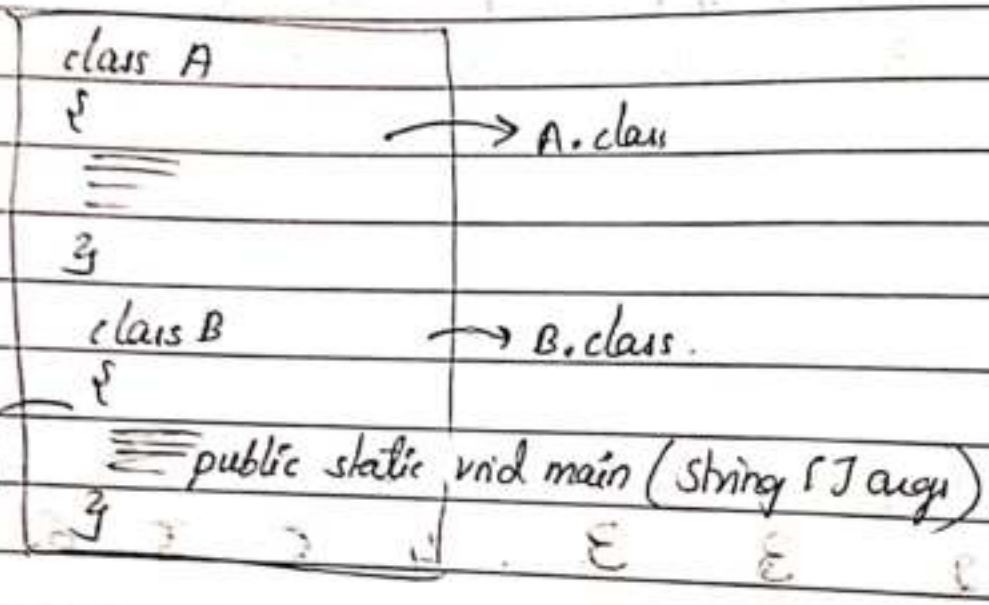
(1)
```
int i=1
i = i++ + i + ++i + i++ + i + ++i + i++ + i + i++
    1     2     3     3     4     5     5     6     6
```

```
class Sample
{
public
 {
 int i=1;
 i = i++ + i++ + i++ + i++ + i + i++ ;
 SOP (i)
 }
}
```

NOTE:

We can develop multiple classes in a single file, which ever class is having main method that class name should be the file name.

For each and every class in the file there will be a corresponding .class file generated.



Example :-

```
class Demo1
{
  static int x = 10;
  static void add()
  {
    SOP("Hello")
  }
}
class Sample1
{
  public static void main(String[] args)
  {
  SOP(" --- Main Start --- ");
  Demo1.add()
  SOP(x);(Demo1.x);
  SOP(" ----- Main End ---- ");
  }
}
```

(1) Write a pgm to calculate area of the circle without returning any value, without parameter between the classes

```
class Tester
{
  static double area()
  {
    int r = 8;
    final double pi = 3.142;
    double result = pi * r * r;  SOP(result);
  }
}
class Demo
{
  public static void main(String[] args)
```

```
    SOP (..... Main Start ....);
    Tester.area();
    SOP (..... Main end .....);
  }
}
```

(3) Write a pgm to calculate area of circle to calculate with method with parameter b/c the classes with methods as static.

```
class Sample2
{
  static void area (int r)
  {          (int r)
  final double pi = 3.142;
     double result = pi * r * r
    return result;
  }
  public static
class Samp3
{
public static void main (String [] args)
{
    area.(2)
    Sample2.area()
```

```
class Sample2
{
static void area(int r)
{
final double pi = 3.142;
    int r;
double result = pi * r * r;
SOP(result);
}
} class Demo1 {
public static void main(String[] args)
{
    Sample2.area(5);
}
}
```

(3) Write a pgm to calculate area of the circle with method with parameter return type both the classes with methods as static

```
class Sample3
{
static double area()
{
final double pi = 3.142;
  int r = 8;
  double result = pi * r * r;
  return result;
}
}
class Demo3
{
public static void main(String[] args)
{
    Sample3.area();
}}
```

```
double x = Sample 3.area();
    sop(x);
}
}
```

pyramid book

PROGRAMMING

1. Write a pgm to print from 1 to 10 using for loop

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

2. Write a pgm to find factorial of a given number

```
class Factorial
{
public static void main (String[] args)
{
    int fact = 1; → because multi start with 1
    for (int i=1; i<=5; i++)
    fact = fact * i
         = 1 * 1
         1 * 2
}
SOP (fact);
}
```

if we put SOP after then only i value is printed one by one and we close the loop and if condition meets then it will be printed

5!
1×2×3×4×5

i = fact

fact = fact * i
= 1 * 1

fact = 1

fact = fact * i → i++
1 * 2
2

6 * 4
24

2 * 3
6

20 * 5
100

Q11

```
class Tester
{
public static void main (String[] args)
{
  int x = 5;
  int fact = 1;
  for (int i = 1; i <= x; i++)
  {
    fact = fact * i;
  }
  System.out.println (fact);
}
```

(1) Write a pgm to reverse a string without using reverse() method

```
class Reverse
{
public static void main (String [] args)
{
String s1 = "Sundra";
for (int i=5 ; i>=0 ; i--)
{
System.out.println(s1.charAt(i));
}
}
```

println (cursor will move to next line)
print (cursor will stay on the same line
a/r/d...

o/p :- ardnuS.

```
class String
{
int length ()  → Full string length will be counted
{
}
Char CharAt (int Index)
{
}
}
```

(6)

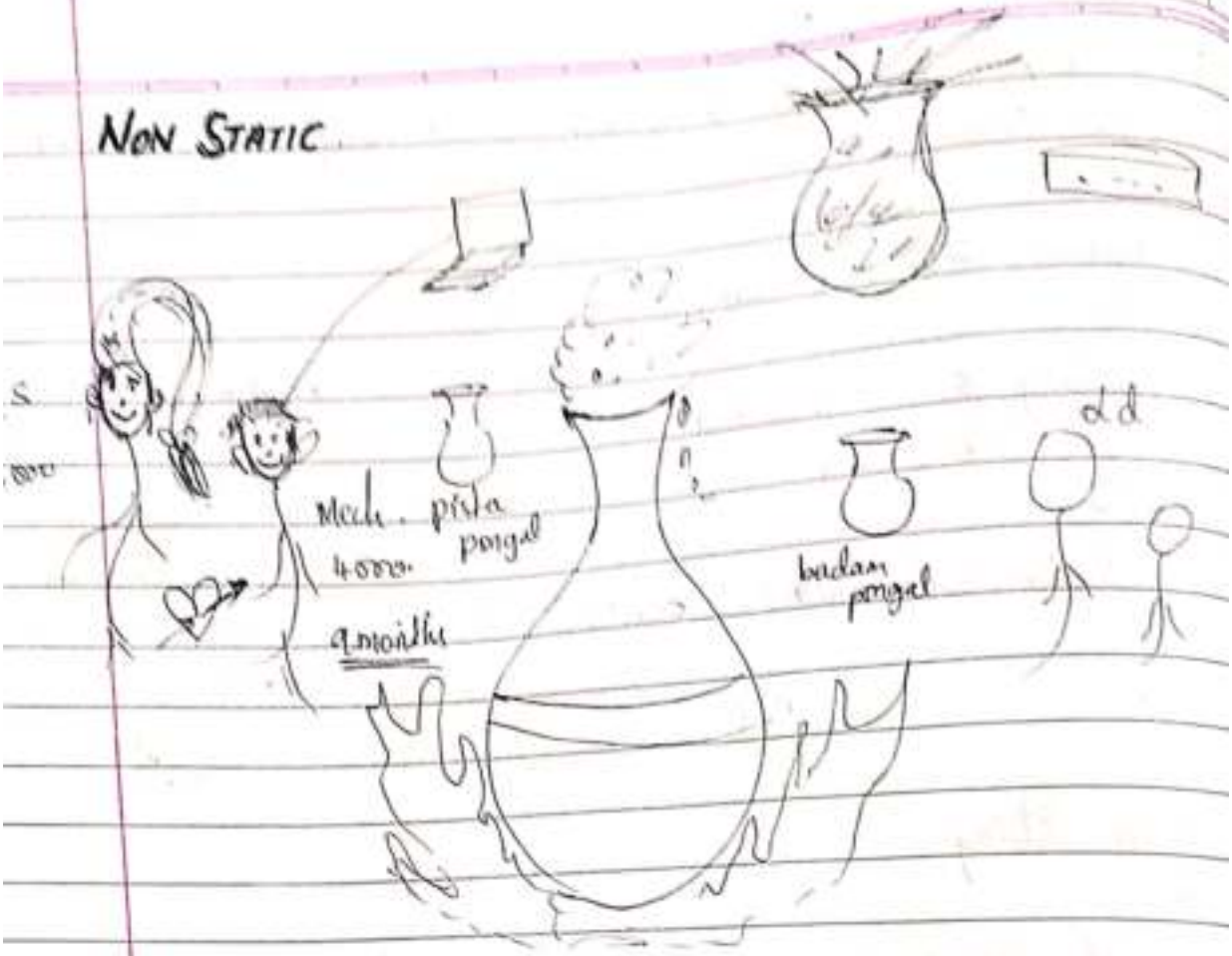
index Sundra
0 1 2 3 4 5

→ to reverse.

(2)

```
class Reverse
{
public static void main (String [] args)
{
String s1 = "Sundra is going for a date";
for (int i = (s1.length)-1 ; i>=0 ; i--)
{
System.out.println (s1.charAt(i));
}
}
}
```

NON STATIC



→ Any member of the class declared without the keyword static is called Non Static member of the class

→ Non Static is always associated with object.

→ Non Static is multiple copies.

→ Whenever we want to access from non-static to static, we have to create object. **i.e.** Object.variable_name or Object.method_name or ref_variable.variable_name or reference_variable.method_name

→ All the non-static members will be stored in "Heap memory".

Page No.:
Date:

Syntax of Object Creation:

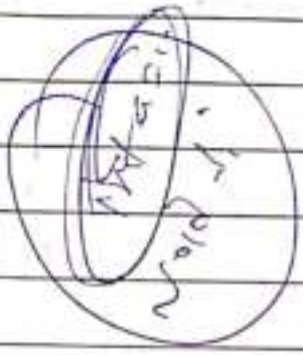
```
new class_name()
```

operator
It will create random memory space into the heap memory

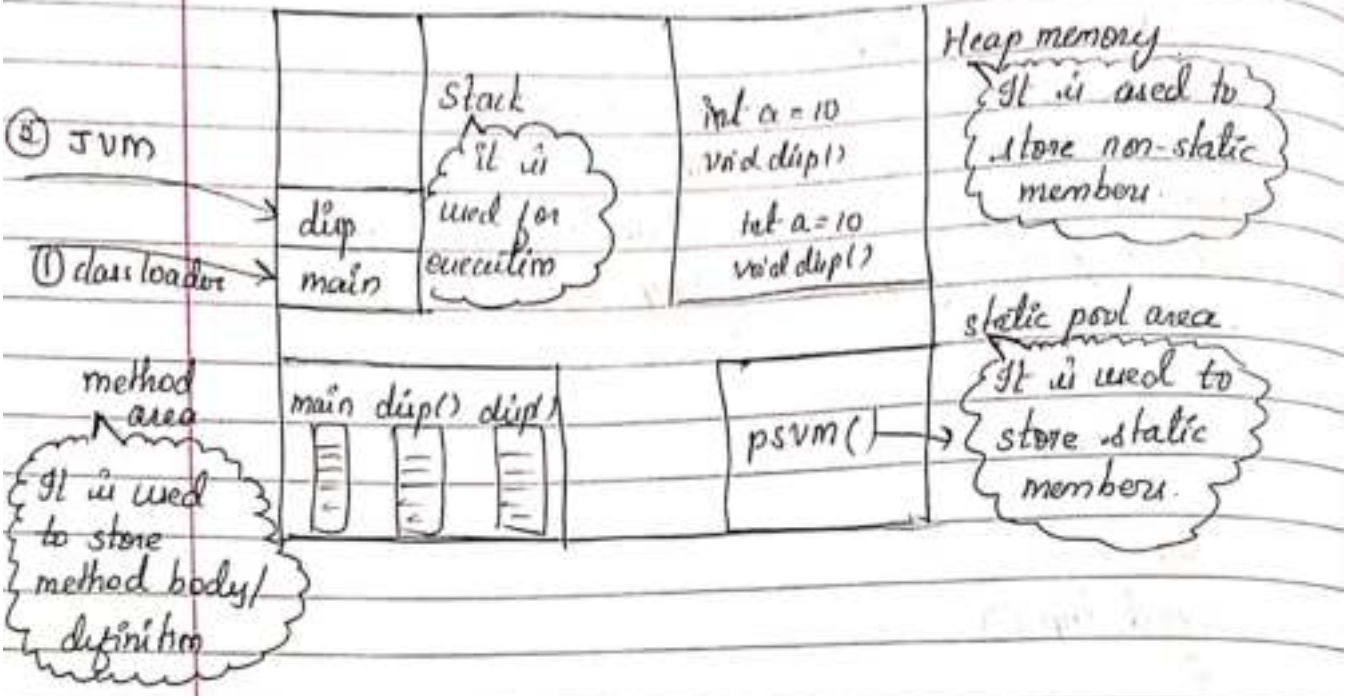
Constructor
constructor will initialize all non-static members into the heap memory

```
class Demo
{
    int a = 10;
    void dup()
    {
        SOP("Hi");
    }
    public static void main(String[] args)
    {
        new Demo().dup();
        SOP(new Demo().a);
    }
}
```

JVM →



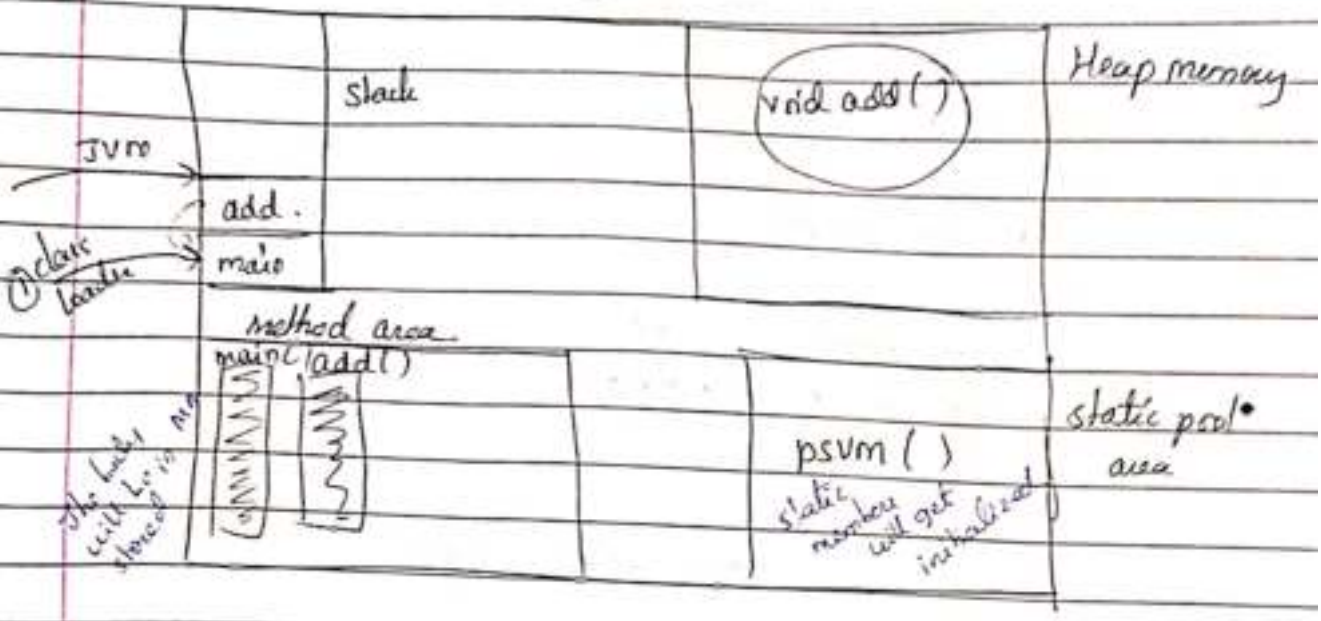
JVM Memory



→ Write a pgm to calculate addition of 2 no's from non static to static.

```
class Addition {
{   void add ()
    int a = 10;
    int b = 5;
    int c = a+b;
    Sop (c);
}
public static void main (String[] args)
{
    new Addition().add ();
}
}
```

JVM Memory



→ Write a pgm to calculate area of the circle from non-static to static

```
class Circle
{
  void area()
  {
     int r = 4;
     final double pi = 3.142;
     double result = pi * r * r;
     SOP (result);
  }
     public static void main (String [] args)
     {
        new Circle().area();
     }
}
```

→ Write a pgm to calculate area of circle with method with parameter from non static to static

```
class Circle
{
  void area (int r)
  {
    final double pi = 3.142;
    double result = pi * r * r;
    SOP (result);
  }
  public static void main (String [] args)
  {
    new Circle (). area (9);
  }
}
```

→ method with return type.

```
class Circle
{
  double area ()
  {
    int r = 5;
    final double pi = 3.142;
    double result = pi * r * r;
    SOP (result);
    return result;
  }
  public static void main (String [] args)
  {
    double a = new Circle (). area ();
    SOP (a);
  }
}
```

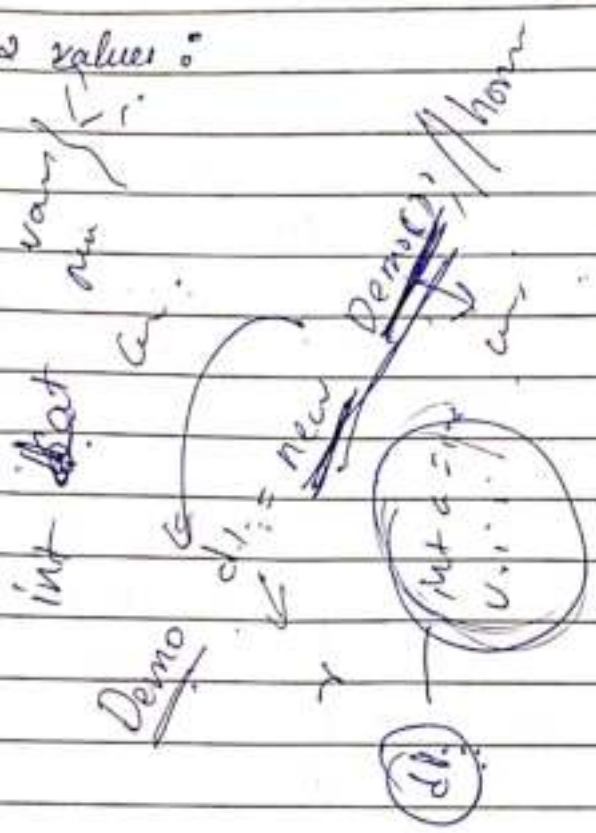
REFERENCE VARIBLE

It is a special type of variable which is used to store object address.

Reference variable can hold 2 values:

i.e. ① Object address

② Null.



Reference Variable Declaration:-

Syntax:

class_name reference_variable

Example:- Demo d1;

Reference Variable Initialization:-

Syntax:

reference_variable = object

Example:- d1 = new Demo();

Combination of reference variable declaration and initialization in single line.

Syntax: class_name reference_variable = new class_name();

Example:- Demo d1 = new Demo(); // homogeneous type of object creation

Demo → class type

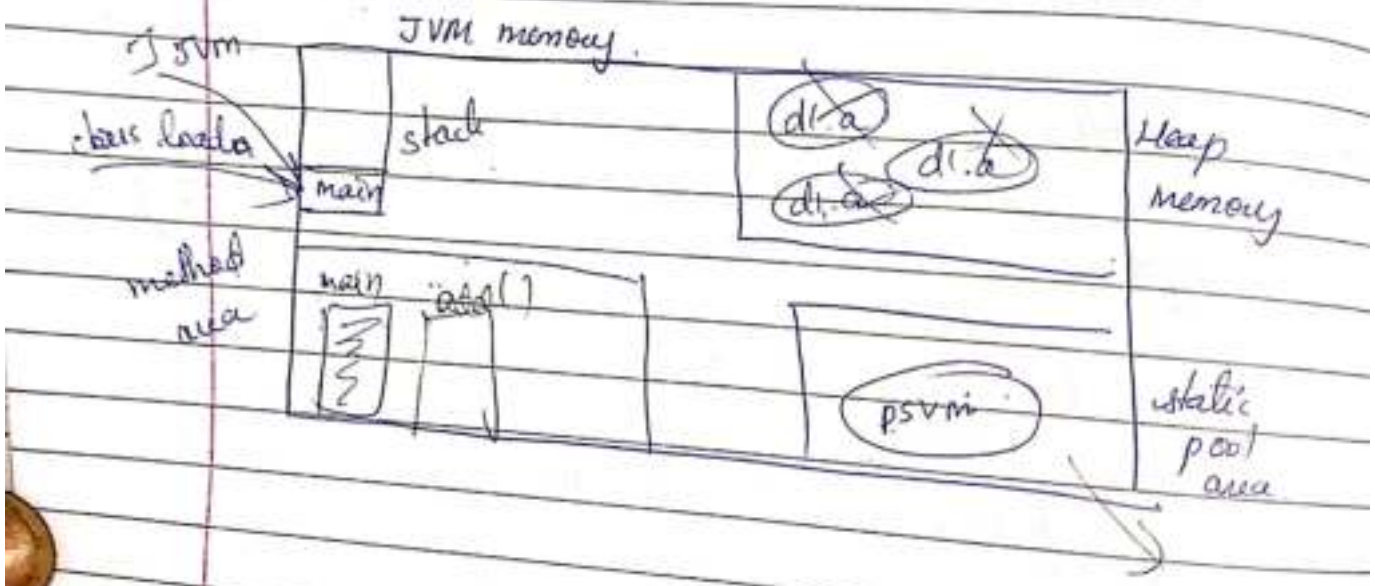
d1 → reference variable

new Demo() → Object

new → operator

Demo() → constructor.

```
class Demo
{
    int a = 10;
public static void main (String[] args)
{
Demo d1 = new Demo();
SOP (d1.a);
SOP (d1.a);
SOP (d1.a);
}
}
```



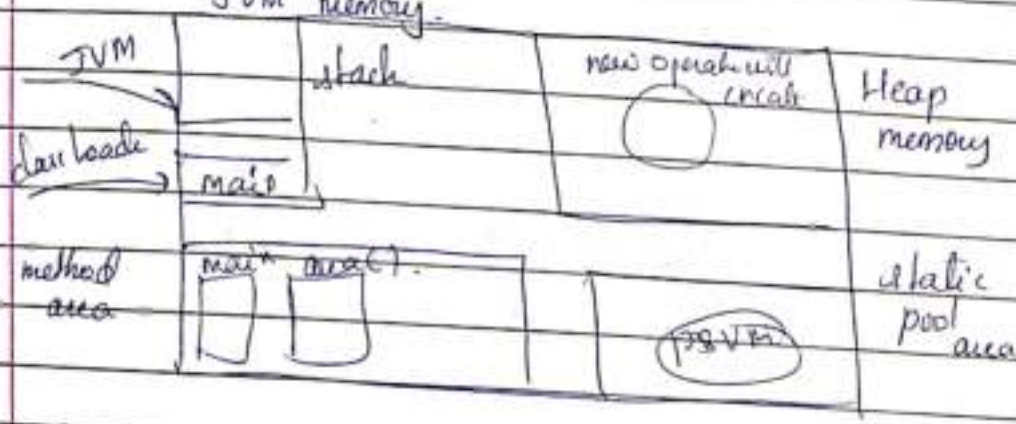
Page No.:
Date:
yourvA

:→ Write a pgm to calculate area of a circle from non static to static through reference variable

use the memory in an efficient way.

```
class Circle
{   void area()
{   int r=5;
    final double pi=3.142;
    double result = pi*r*r; }
public static void main (String [] args)
{
    Circle d2 = new Circle();
    SOP( d2.area); d2.area();
}
```

holds → reference variable




```
class Circle
{
    void area()
    {
        int r=5;
        final double pi=3.142;
        double result = pi*r*r;
        SOP(result);
    }
    PSVM(--)
    {   Circle c1 = new Circle();
        c1.area(); }
```

→ Write a pgm to calculate area of a circle with method with parameter through reference variable

```
class Circle
{
void area (int r)
{
    final double pi = 3.142;
    double result = pi * r * r;
    SOP (result);
}
public static void main (String [] args)
{
        Circle d2 = new Circle ();
            d2. area (5);
}
}
```

→ Write a pgm to calculate area of a circle with method with return type through reference variable.

```
class Circle
{
    double area ()
    {
        int r = 5;
        final double pi = 3.142;
        double result = pi * r * r;
        return result;
    }
    public static void main (String [] args)
    {
            Circle d3 = new Circle ();
        double x = d3. area ();
```

Page No.:
Date:

```
SOP(k);
}
}
```

Real time example:

Tanavi and Rama joined Dspiders. They started with course Java. In 1st mock Tanavi got mark scaling 1 and Rama got 2 and Rama got depressed and he spoke to Shri BF Jamuna. Jamuna suggested to take remock with some spirit. Rama studied day and night for the remock and got the mark scaling as 1. Write the program for the scenario and print Rama and Tanavi mock scaling.

If we create multiple object each object will have unique address

```
class Sample
{
  public static void main(String[] args)
  {
  Sample s1 = new Sample();
    SOP(s1);
  Sample s2 = new Sample();
    SOP(s2);
```

Whenever we print a reference variable it will print the object address.

Object address is called as fully qualified path.

```
class Sample
{
  public static void main(String[] args)
  {
  Sample s1 = new Sample();
    SOP(s1);
  Sample s2 = s1;
    SOP(s2);
  }
}
```

```
class Aspiderx
{
  int Javamock
  {
  public static void main (String[] args)
  {
  Aspiderx std1 = new Aspiderx()
   std1.Javamock = 1;
  SOP (std1.Javamock);
  Aspiderx std2 = new Aspiderx()
    std2.Javamock = 2;
     SOP (std2.Javamock)
     std2.Javamock = 1;
    SOP (std2.Javamock);
}
}
}
```

```
class Demo
{
static String brand = "Dell";
    int price;
    String colour;
public static void main (String[] args)
  {
    Demo d1 = new Demo();
      d1.price = 90,000;
      d1.colour = 'Black';
      SOP (d1.price);
      SOP (d1.colour);
      SOP (brand);
  }
}
```

→ Write a pgm to add 2 numbers from non static to static between the classes through reference variable.

```
class Sample
{
  void add ()
{
    SOP (10+20);
}
}
class Demo.
{
  public static void main (String[] args)
    {
      Sample s1 = new Sample ();
        s1.add ();
  }
}
```

6. Write a pgm to calculate area of a circle from non static to static between the classes.

```
class Circle1
{
    void area()
    {
        int r = 1;
        final double pi = 3.142;
        double result = pi * r * r;
        SOP(result);
    }
    ~~public~~
}
class Circle2
{
    public static void main(String[] args)
        Circle1 c1 = new Sample();
            c1.area();
    }
}
```

7. Write a pgm to calculate area of a circle from non static to static with method with parameter.

```
class Circle1
{
    void area(int r)
    {
        final double pi = 3.142;
        double result = pi * r * r;
        SOP(result);
    }
}
```

```
class Circle2
{
public static void main (String[] args)
{
Circle1 c1 = new Circle1 ();
    c1. area (5);
}
}
```

→ write a pgm to calculate area of a circle from non-static to static with methods return type b/w the class

```
class Circle1
{
  double area ()
  {
    int r = 6;
    final double pi = 3.142;
    double result = pi * r * r;
    return result;
  }
}
class Circle2
{
  public static void main (String[] args)
  {
    Circle1 c1 = new Circle1 ();
double r r = c1. area ();
    SOP (r);
  }
}
```

BLOCK

Java provides a separate block called static initialization block (SIB), Instance Initialization Block (IIB)

SIB

→ Any Block which is declared with a key word static is called as static initialization block

→ SIB will get executed before main method

Syntax:

```
static {
    ==========
}
```

Example:

```
class Demo
{
static {
        SOP("SIB1");
    }
static {
        SOP("SIB2");
    }
public static void main (String[] args)
{
    SOP("Hi");
}
}
```

IIB

→ Any Block which is declared without the keyword static is called as Instance Initialization Block.

→ IIB will get executed whenever an object is created

Syntax:

```
{
  ==========
}
```

Example:

```
class Sample
{
  {
    SOP ("IIB1");
  }
  {
    SOP ("IIB2");
  }
public static void main (String[] args)
  {
    SOP ("---- Main start ----");
      new Sample ();
    SOP ("---- Main End ----");
  }
}
```

→ What is the difference between Static and Non-Static ?

	Static	Non-Static
(1)	Any member of the class declared with the keyword static is called as Static member of the class	Any member of the class declared without the keyword static is called as non-static member of the class
(2)	Static is always associated with class	Non-Static is always associated with object.
(3)	Static is one copy	Non-static is multiple copies.
(4)	All the static members will be stored in 'static pool area'.	All the non-static members will be stored in 'heap memory'.
(5)	Whenever we want to access static member from 1 class to another class we have to use classname.variable_name or class_name.method_name.	Whenever we want to access from non-static to static we have to go for object.method_name or object.variable_name or reference_variable.variable_name or ref_variable.method_name
(6)	Java provides a seperate block called SIB i.e. Static Initialization Block	Java provides a seperate block called IIB i.e. Instance Initialization block

① SIB will get executed before main method.	IIB will get executed whenever an object is created.
② We can have 'n' no. of SIB's. The order of execution is sequential.	We can have 'n' no. of IIB's The order of execution is sequential
③ Only variable and methods can be declared as static and non static	Only constructor can be declared as non-static and it can never be static

CONSTRUCTOR

It is a special method or special member of the class which is used to initialize data members

RULES :-

1] The constructor name should be same as class name.
2] Constructor will not has return type.
3] Constructor will not return any value.
4] Constructor is always non static.
5] Whenever an object is created constructor will get invoked.

Syntax :

```
class class_name
{
    class_name ()
    {
    ==
    return;
    }
}
```

Example :

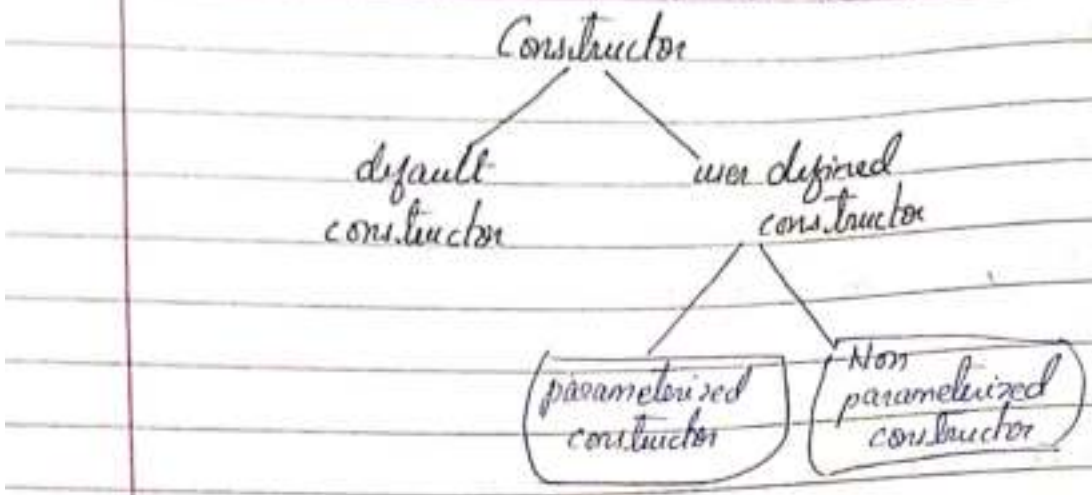
```
class Demo
{
    Demo ()
    {
    ==
    }
}
```

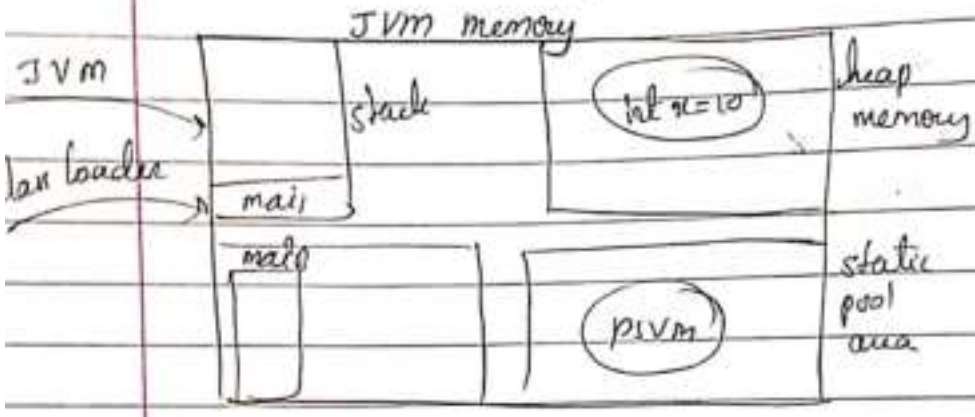
→ constructor

```
class Sample.
{
  Sample ()
  {
    SOP("Hey I am constructor");
    return;
  }
public static void main (String[] args)
  {
    SOP(-----Main start-----);
    new Sample ();
    SOP(-----Main End-----);
  }
}
```

→ Write a program to initialize the variable through constructor.

```
class Sample
{
  int x;
  Sample (int y)
  {
    x=y;
  }
public static void main (String[] args)
  {
    Sample S1 = new Sample (10);
    SOP (S1.x);
  }
}
```

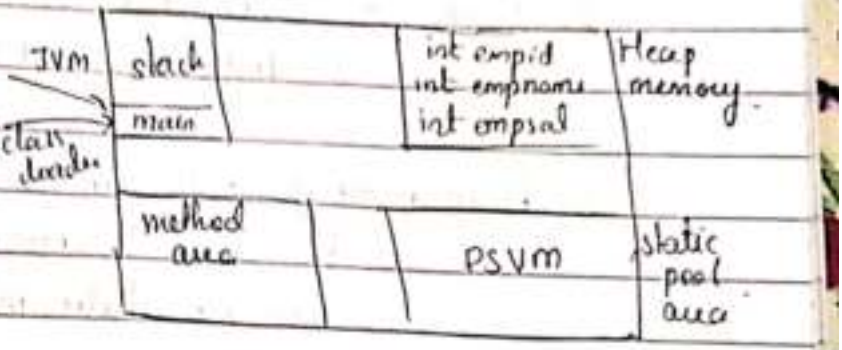




→ Write a pgm to initialize employee ID, employee name and employee salary through constructor.

```
class Employee
{
    int empid;
    String empname;
    double empsal;
    Employee (int x, String y, double z)
    {
        empid = x;
        empname = y;
        empsal = z;
    }
    public static void main (String[] args)
    {
```

```
Employee e1 = new Employee (103, "Dinga", 20000-56);
    SOP(e1.empid);
    SOP(e1.empname);
    SOP(e1.empsal);
}
}
```



Whenever a class loader loads the class all the static members will get initialized in the static pool area.

JVM starts executing from main method in the above pgm the first statement is the object creation, equal operator will work from right to left.

New operator will create a random memory space in the heap memory. Constructor will initialize all the non-static members into the heap memory while creating the object itself we pass the arguments which will get initialized the constructor and from constructor it will get initialized to the object variable.

Now in the main method the object address will be stored in the reference variable e1 and through that reference variable we print the values

JVM MEMORY

It has 4 parts

1. Stack → It is used for execution which follows last in first out [LIFO]

2. Heap memory → It is used to store non-static members of the class. Whenever we create an object of the instance non static members will be stored in heap memory.

3. Static Pool Area → It is used to store static member of the class. Whenever the class loads the class at that instance static members will get initialized into the static pool area.

4. Method area → It is used to store method body or definition. Irrespective of static or non-static all the method bodies will be stored in method area.

→ WAP to initialize studentid, student name and student fees through constructor.

```
class Student
{
 int student_id;
 String student_name;
 double student_fees;
 Student (int a, String b, double c)
 {
  student_id = a;
  student_name = b;
  student_fees = c;
 }
```

```
public static void main (String[] args)
{
  Student S1 = new Student ("Sallu", 133, 40000.85)
   S.O.P (S1.student_id);
   S.O.P (S1.student_name);
   S.O.P (S1.student_fees);
}
}
```

THIS KEYWORD

It is used to perf point to the current object whenever the local variable and global variable names are same to differentiate between them we use "this" keyword.

→ This keyword should be used only in the non static const.
→ This keyword is the default reference variable.

```
class Demo
{
 int a=10;
 void add()
 {
   int a=90;
   S.O.P(a);
   S.O.P(this.a);
 }
}
```

```
class Employee
{
int empid;
String empname;
double empsal;
Employee (int empid, String empname, double empsal)
{
 this.empid = empid;
 this.empname = empname;
 this.empsal = empsal;
}
public static void main (String [] args)
{
Employee e1 = new Employee (123, "Sallu", 30000.00);
 S.O.P (e1.empid
 s.o.p (e1.empid);
 s.o.p (e1.empsal);
}
}
```

→ WAP to initialize company name, salary, location through constructor using this keyword.

```
class Company
{
    String compname;
    double sal;
    String loc;
    Company (String compname, double sal, String loc)
    {
        this.compname = compname;
        this.sal = sal;
        this.loc = sal;
    }
```

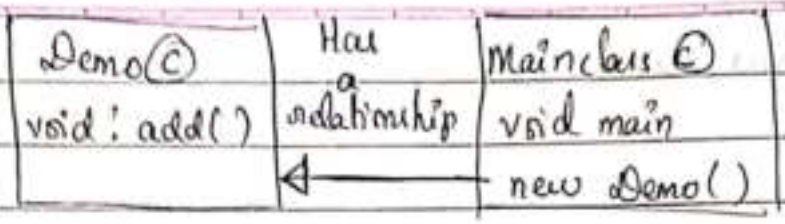
```
public static void main (String [] args)
{
    Company c1 = new Company ("TCS", 30000.00, "Banglore")
    S.O.P (c1.compname);
    S.O.P (c1.sal);
    S.O.P (c1.loc);
}
}
```

COMPOSITION :

It is also called "Has a relationship".
A class having an object of another class is called as Composition.

Class diagram :-
It is a pictorial representation to represent the members of the class.

```
class Demo
{
  void add ()
  {
    S.O.P ("Hi");
  }
}
  class Mainclass
{
  public static void main (String [] args)
  {
    Demo d1 = new Demo ();
        d1.add ();
}
}
```



METHOD OVERLOADING:

Developing multiple methods with the same name but variation in arguments list is called as "Method Overloading".

Variation in argument list means:
1] Variation in the data type.
2] Variation in the length of the argument.
3] Variation in the order of occurance of the argument

Rules:
1] The method name should be same.
2] There should be variations in argument list.
3] There is no restriction on access specifier, modifier and return type.

NOTE:→
→ We can overload both static and non-static method
→ We can't overload main method

(1)
```
class Whatsapp
{
    void send(int no)
    {
        SOP("sending no" + no);
    }
```

```
void send (string msg)
{ SOP ("Sending text msg" + msg);
}
void send (int no, String msg)
{
 S.O.P ("send no & msg" + no + " " + msg);
}
void send (String msg, int no)
{
 S.O.P ("sending msg & no" + msg + " " + no);
}
}
class Mainclass
{
public static void main (String [] args)
{
Whatsapp wa = new Whatsapp();
 wa.send (123);
 wa.send ("Hello");
 wa.send (106, "Hi");
 wa.send ("Bye", 127);
}
}
```

(2)
```
class Bookmyshow
{
static void book (int no_seat)
{ S.O.P ("book by no. of seats");
} static void book (String movie_name)
{ S.O.P ("book by movie name");
}
static void book (int no_seat, String
        movie_name)
{
 SOP ("book by no. of seats & movie
        name");
} static void book (String movie_name,
        int no_seats)
{
SOP ("book by movie name & no
        of seats");
} psvm (- - - -)
{
 book (5);
 book ("bandey");
 book ("banatey", 2);
 book (4, "banatey");
}
}
```

CLASS AND OBJECT

Class is a blue print or a template to create object.

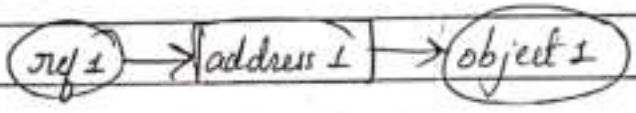
What is object?

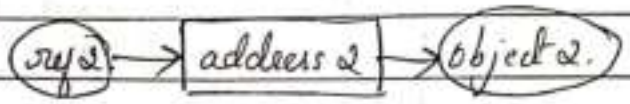
Object is a real time entity which has its own state and behaviour.

→ State defines the non static variables, what data it can hold.

→ Behaviour defines non static methods, and the way it can behave.

* Whenever we create an object we get both state and behaviour.
* The objects address will be stored in reference variable.
* Multiple reference variables can hold multiple object object address.

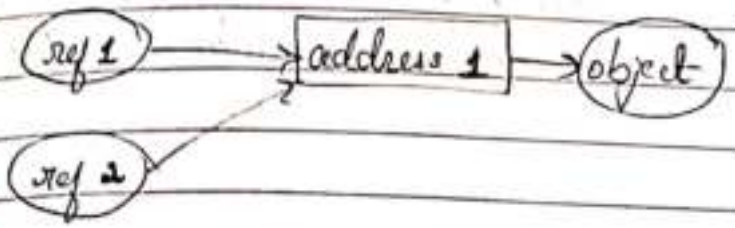




* Any changes made to the object through a reference variable, it will not affect other reference variable.

* Multiple reference variable can point (hold) single object address.

* Any changes made through a reference variable will affect other reference variable



PASS BY VALUE

Calling or invoking a method by passing primitive type of data is called as 'Call by Value' or 'Pass by Value'

Example:

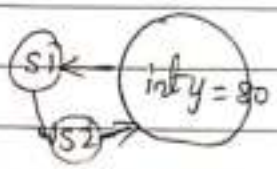
```
class Sample
{
static void add (int a)
 {
  SOP(a);
 }
public static void main (String[] args)
 {
   int x = 10;
   add (x);
 }
}
```

PASS BY REFERENCE

Calling or invoking the method by passing reference variable is called as 'pass by reference'

Example:

```
class Sample
{
    int y = 80;
    static void cool (Sample s2)
    {
        SOP (s2.y);
    }
    public static void main (String [] args)
    {
        Sample s1 = new Sample ();
        SOP (s1.y);
        cool (s1);
    }
}
```



int y = 8
cool ( )

```
class Wipro aunty
{
 void testengg()
 {
   S.O.P("Opening for Testing job");
 }
}
   class uncle
   {
    public static void main(String[] args)
   {
    WiproAunty a1 = new Wiproaunty();
    Dinga.need job(a1);
    Dinga.need job(a1);
   }
    class Dinga
   {
    static. void needjob(wiproAunty.a1);
    {
     a2.testengg();
    }
   }
    class Dinga
   {
    static void need job(wiproAunty.a2);
    {
      a3.testengg();
    }
   }
```

ARRAY

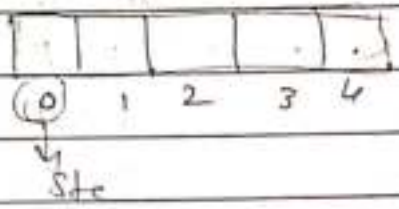
Array is an linear data structure which is used to store homogenous (same) type of data.

→ In array the size is fixed and it can store only homogeneous type of data. There are the drawbacks of an array.

Array

```
int a = 10;
int b = 20;
int c = 30;
int d = 40;
int e = 50;
```

```
int a = 10;
int b = a;
```

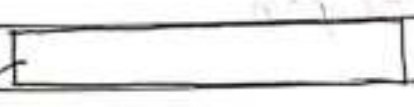


Array Declaration

Syntax:

```
datatype [] array_name;
```

Example: int [] arr



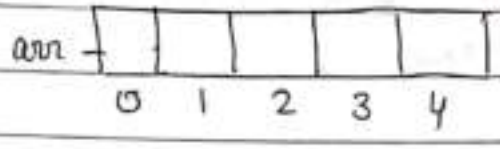
Array Size Initialization

Syntax:-

```
array_name = new datatype [size];
```

Example :-

arr = new int [5];



Store the value into an Array

Syntax :-

array_name [index] = value ;

```
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

Array Initialization

```
S.O.P (arr[0]);
S.O.P (arr[1]);
S.O.P (arr[2]);
S.O.P (arr[3]);
S.O.P (arr[4]);
S.O.P (arr[5]);
```

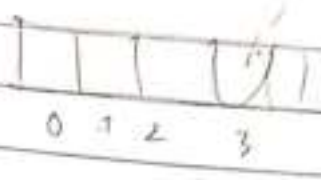
O/P = 10
20
30

Array Index Out Of Bound Exception

Array value Re-Initialization

Syntax:- array_name[index] = new value;

Example:- arr[3] = 90;



If the value is not stored in a particular index and if we try to print, we will get default values.

To find length of an Array

Syntax:- array_name.length;

arr[0] = 10

arr.length = 5

S.O.P (arr.length); o/p = 5

```
for (int i=0; i<arr.length; i++)
                0<5
                1<5
                2<5
                3<5
{
    S.O.P (arr[i]);
}
```

Array Declaration and store the value Directly

Syntax:-

datatype[] arrayname = {v1, v2, v3, v4};

Example:-

int[] abb = {10, 20, 30, 40, 50};

abb →	10	20	30	40	50
	0	1	2	3	4

Copying the value from one Array to another Array

```
int [ ]acc = abb;
```

acc	10	20	30	40	50

```
class Sample1
{
  public static void main (String[] args)
  {
    char[] arr = new char[3];
    arr[0] = 'A';
    arr[1] = 'B';
    arr[2] = 'C';

    S.O.P ("-------");
    S.O.P ("index \t values");
    S.O.P ("-------");
    for (int i=0; i< arr.length; i++)
    {
      S.O.P (i + "\t" + arr[i]);
    }
  }
}
```

```
class Sample2
{
public static void main (String[] args)
{
boolean [] arr = { true, false, true, false};
for (int i=0; i<arr.length; i++)
{
  S.O.P ( i + "\t" + arr[i]);
}
}
}
```

INHERITANCE

Java Inheritance

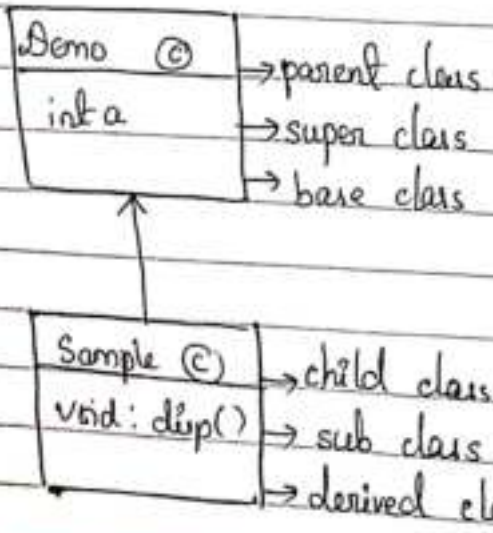
Inheriting the property from one class to another class is called as Inheritance

In Inheritance we have 5 types:

1) Single Level Inheritance.
2) Multi Level Inheritance.
3) Hierarchical Inheritance
4) Multiple Inheritance.
5) Hybrid Inheritance.

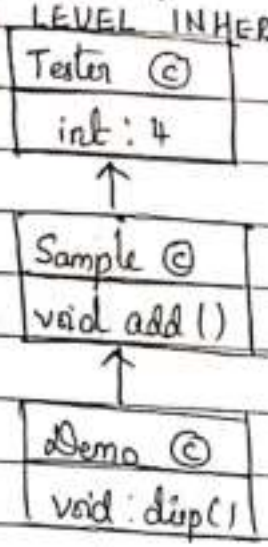
SINGLE LEVEL INHERITANCE

A subclass inheriting the properties from only super class is called as 'SINGLE LEVEL INHERITANCE'.



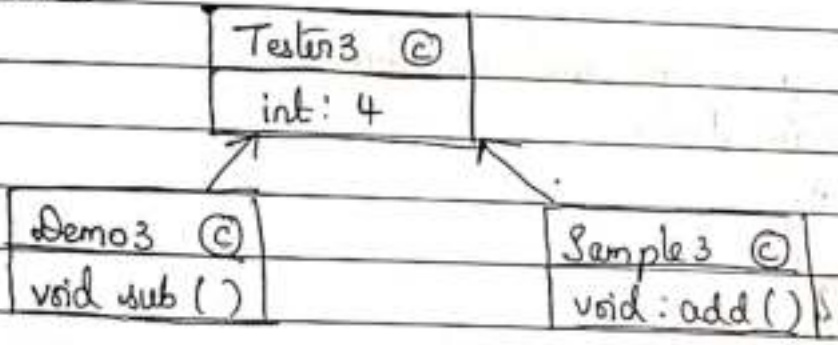
MULTI LEVEL INHERITANCE

A subclass inheriting the properties from its super class intuen super class inheriting the properties from its super class is called as 'MULTI LEVEL INHERITANCE'.



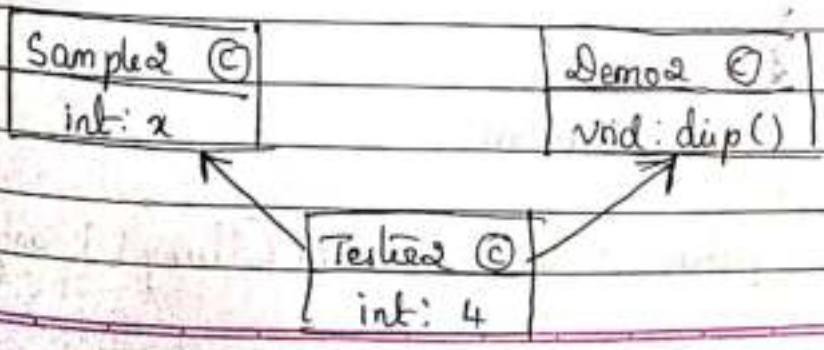
HIERARCHICAL INHERITANCE

A multiple sub class inheriting the properties from only one super class or common super class is called as 'HIERARCHICAL INHERITANCE'.



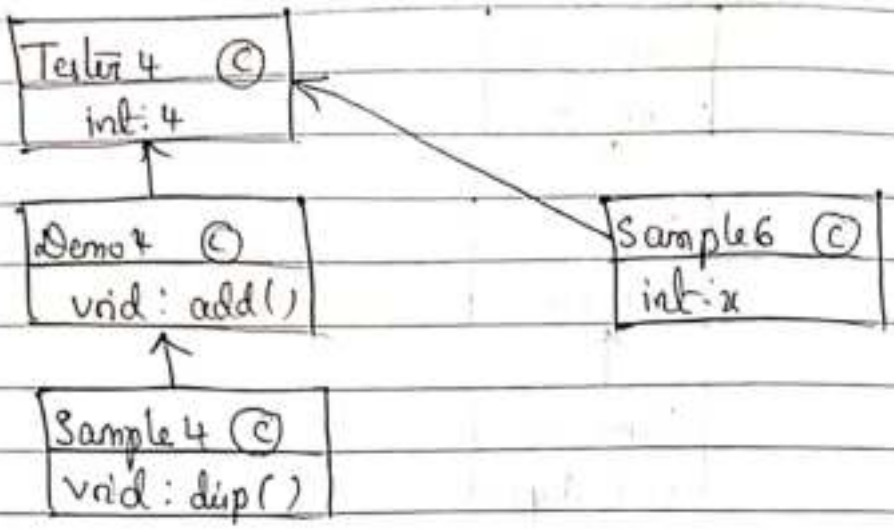
MULTIPLE INHERITANCE

A sub class inheriting the properties from multiple super class.



HYBRID INHERITANCE

It is a combination of single level, multi level and hierarchial inheritance.



CLASS DIAGRAM

It is a pictorial representation to represent like number of the class.

→ Example for single level inheritance.

```
class Demo
{
int a = 10;
}
class Sample extends Demo
{
void disp ()
{
S.O.P ("hello");
}
}
class Mainclass
{
public static void main (String [] args)
```

```
{
  S.O.P ("main start");
  Sample s1 = new Sample ();
      s1.disp ();
      SOP (s1.a);
      SOP ("main end");
  }
}
```

Example for Multilevel inheritance

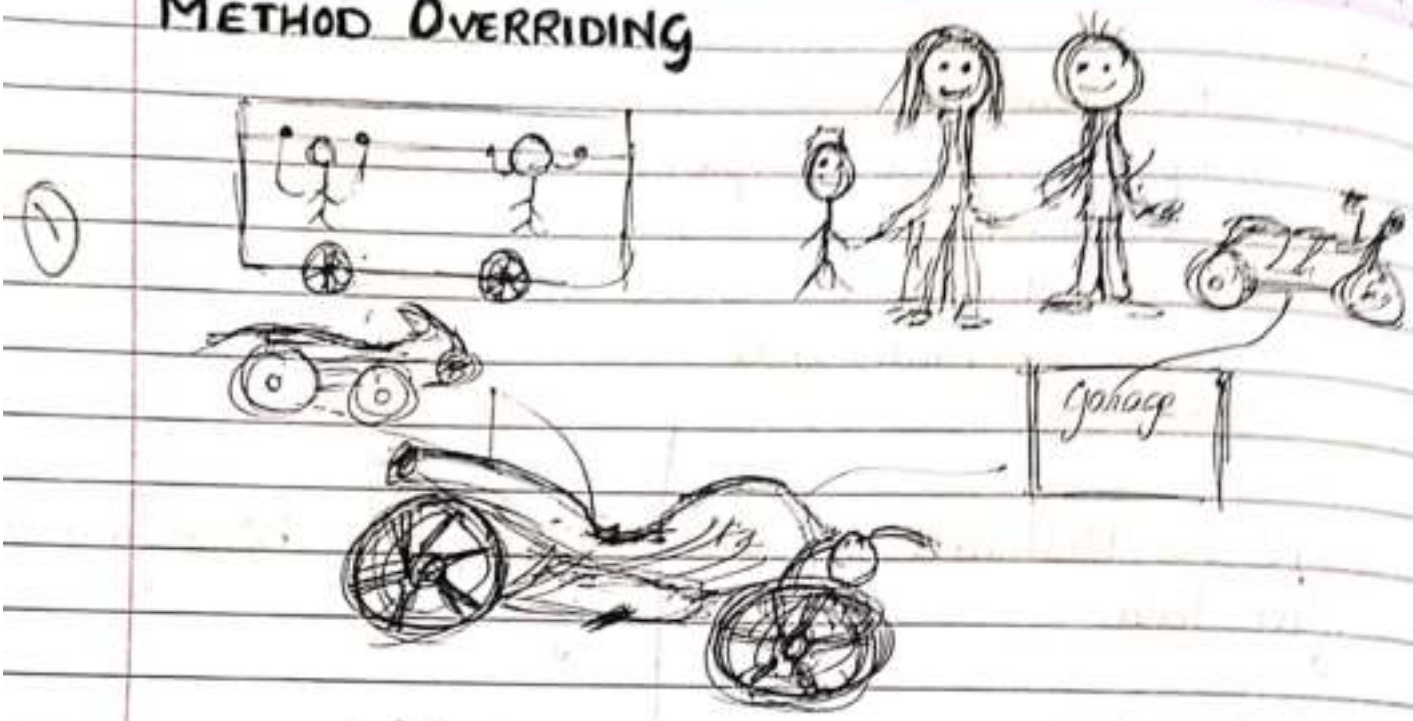
```
class Tester
{
  int y = 10;
}
class Sample extends Tester
{
  void add()
    SOP ("Hi");
}
class Demo extends Sample
{
  void disp()
  {
    SOP ("Hello");
  }
}
class Main_class
{
  public SVM (_ _)
  {
  SOP ("main start");
  Demo d1 = new demo ();
      d1.disp ();
      d1.add();
      SOP (d1.y); SOP ("end"); }}
```

Example for Hierarchical inh.

```
class Tester
{
  int y = 10;
}
class Demo extends Tester
{
  void add ()
  {
  SOP ("Hi");
} }
class Sample extends Tester
{
  void add ()
  {
    SOP ("Hi");
} }
class Mainclass
{
  PSVM (String[] args)
{
  SOP ("main start");
  SOP ("Demo object");
  Demo3 d3 = new Demo();
    } }
```

only non static methods can be inherited
we can overload both static & non static

METHOD OVERRIDING



Developing a method in the sub class with the same name and signature as in the superclass but with different implementation in the subclass is called as "Method Overriding"

RULES

1. The method name and signature should be same in the sub class as in the super class.
2. There should be IS A RELATIONSHIP (inheritance)
3. The method should be non static

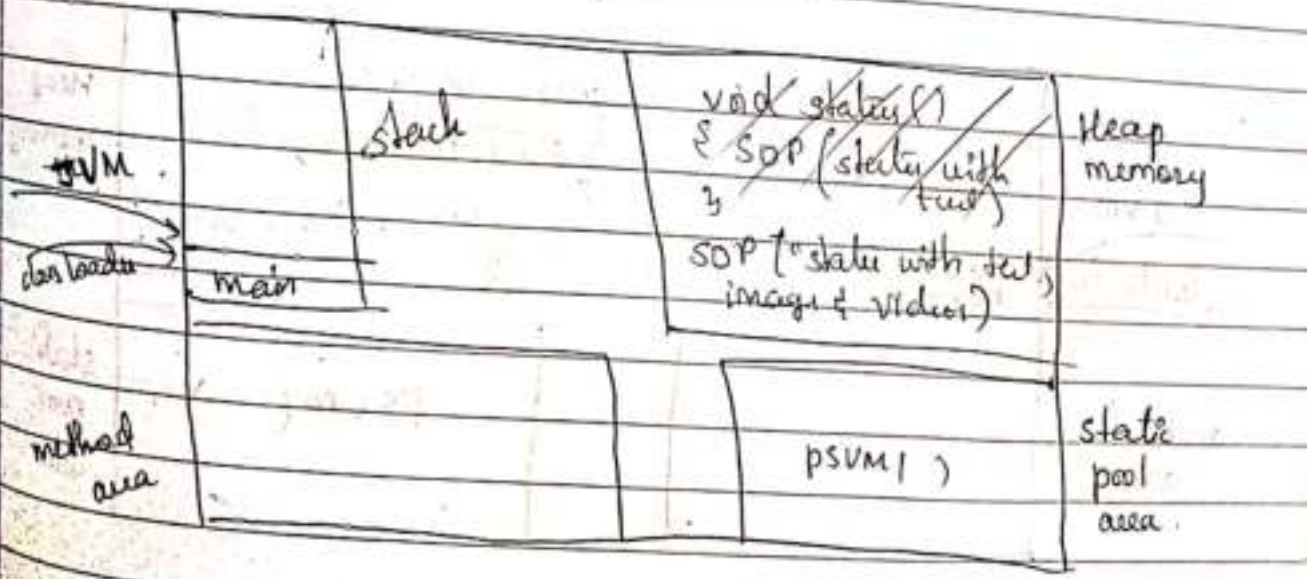
Example:

```
class Whatsapp_v1
{
  void status()
  {
  SOP ("Status with text")
  }
}

class Whatsapp_v2 extends Whatsapp_v1
{
  void status()
  {
    SOP ("Status with text, images & videos");
  }
}.

class Mainclass
{
public static void main (String[] args)
{
Whatsapp w2 = new Whatsapp_v2();
    w2.status();
```

2

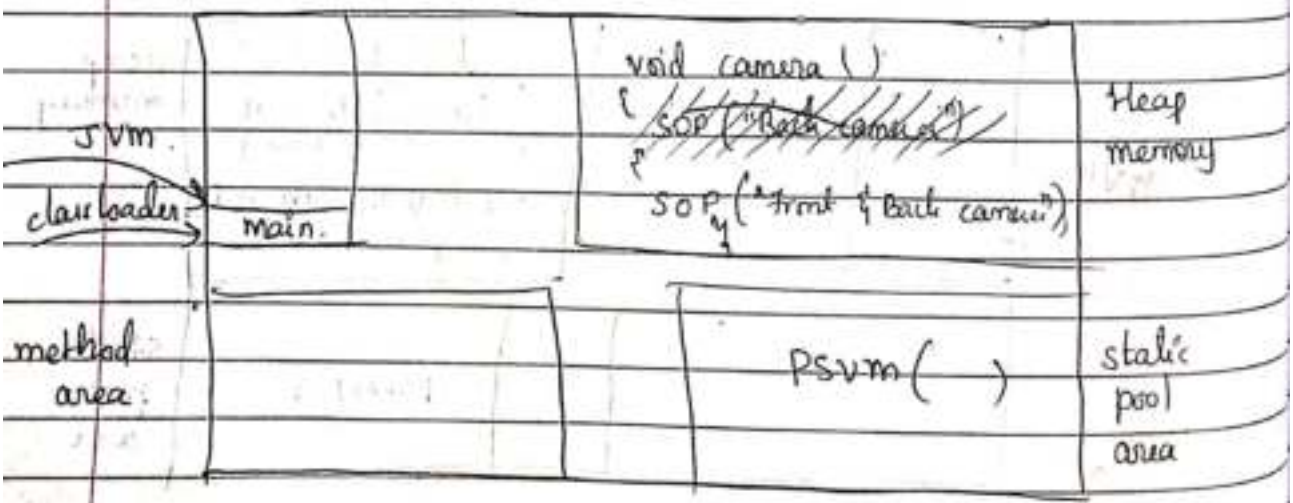
o/p :- status with text, images & videos

(we will get our wished output)



Example:

```
class kitkat
{
 void camera()
 {
  SOP("Back Camera");
 }
}
class lolipop extends kitkat
{
 void camera()
 {
  SOP("Front & Back camera");
 }
}
  class Mainclass
  {
  public static void main(String[] args)
 {
  Lolipop l1 = new lolipop();
       l1.camera();
```



SUPER KEYWORD :

In the case of method overriding, along with the subclass implementation, if we need super class implementation then we should go for super.method name

NOTE

→ We go for inheritance for code reusability

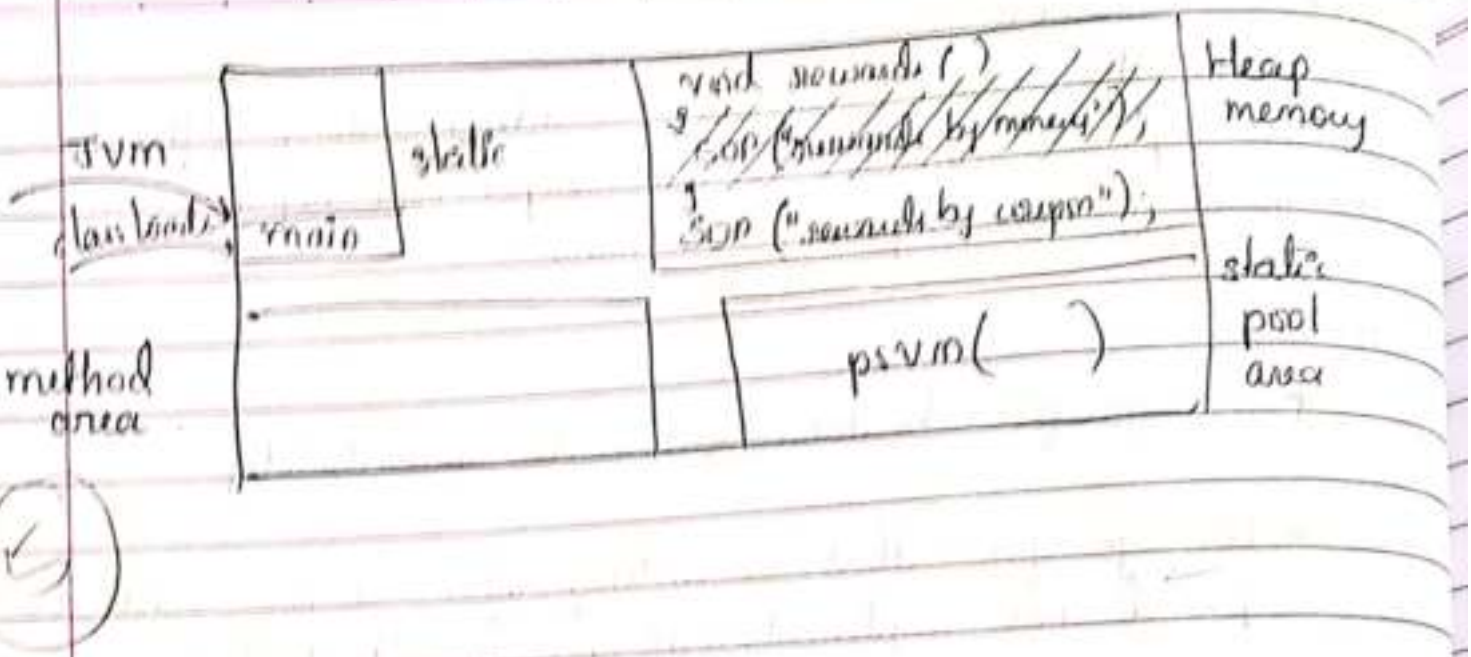
→ We go for method overriding whenever we want to provide new implementation for the old feature.

Example :

```
class phonepe_v1
{
void rewards()
{
SOP ("~~keyword~~ rewards by money");
}
}
class phonepe_v2 extends phonepe_v1
{
void rewards()
{
SOP ("rewards by coupon");
super.rewards();
}
}
class Mainclass
{
public static void main (String[] args)
{
phonepe_v2 p2 = new phonepe_v2();
p2.rewards();
}
}
```

o/p:

rewards by coupon
rewards by money



implicitly means its having internally as: double x = 20 ; impli.
externally or explicitly as: double x = (double) 20;

TYPE CASTING

Converting from 1 type to another type is called as 'TYPE CASTING'

① Primitive Type Casting
- Narrowing → Explicitly
- Widening → Implicitly
  → Explicitly

② Class Type Casting / Derived Type Casting.
- upcasting → implicitly → explicitly.
- down casting → explicitly.

→ PRIMITIVE TYPE CASTING :-

Converting from 1 primitive data type to another primitive data type is called as 'Primitive Type Casting'.

- Widening :-

Converting from smaller primitive data type to any of its bigger primitive data type is called as "widening"

Widening can be done both implicitly and explicitly.

Example :-

	implicitly	explicitly
①	double x = 20;	double x = (double) 20;
	SOP(x); o/p 20.0d.	S.O.P(x); o/p 20.0d.
②	double y = 36.6f;	double y = (double) 36.6f;
	SOP(y); o/p 36.6d.	SOP(y); o/p : 36.6d.
③	float z = 40;	float z = (float) 40;
	SOP(z); o/p 40.0f	SOP(z); o/p 40.0f

- Narrowing :-

Converting from Bigger primitive data type to any of its smaller primitive data type is called as "Narrowing"

Narrowing should be always done explicitly.

Example :-

① long a = (long) 26.6d;
SOP (a); o/p 26.

② int y = (int) 59.9d;
SOP(y); o/p 59.

③ short z = (short) 56.6f;
SOP (z); o/p 56

④ byte c = (byte) 60.6f;
SOP(c); o/p 60.

Example :

```
class Sample1
{
public static void main (String[] args)
{
    SOP(------ widening ----);
    double a = 20; // implicitly widening
    SOP (a);

    double b = (double) 20; // explicitly widening
    SOP (b);

    SOP(------ narrowing ----)
    int y = (int) 20.56; // explicitly narrowing
    SOP (y);
}
}
```

once it is operated we will get only super class objects.

CLASS TYPE CASTING / DERIVED TYPE CASTING :

UP CASTING

Converting from subclass object to super class type is called as 'UP CASTING'.

→ Up Casting can be done both implecitly and, explicitly.

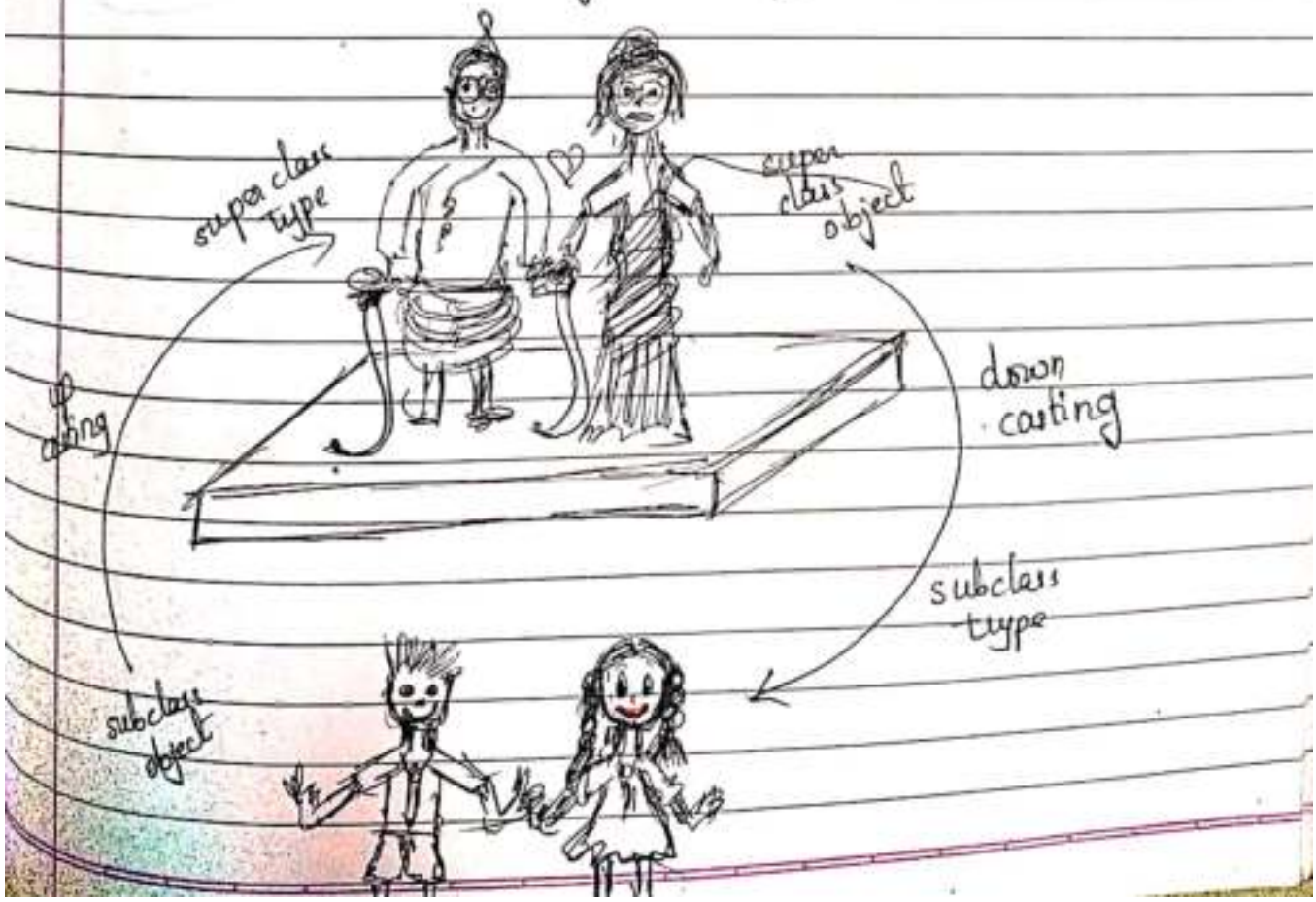
DOWN CASTING

Converting from super class object to sub class type is called as "DOWN CASTING"

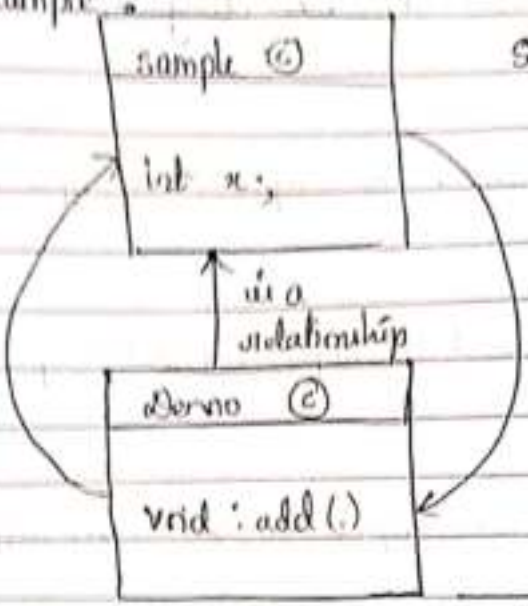
→ Down casting should be done always explicitly

→ Without performing up casting we cannot perform down casting

→ Direct down casting is not possible.



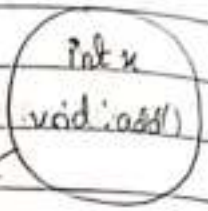
Example :-



Heterogeneous type of object

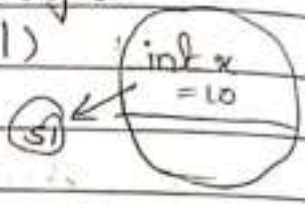
```
Sample S1 = new Demo()
SOP (S1.x);
S1.add();
```


```
Demo d1 = (Demo) S1;
SOP (d1.x);
d1.add();
```

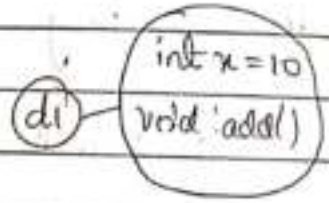


Homogeneous type of object.

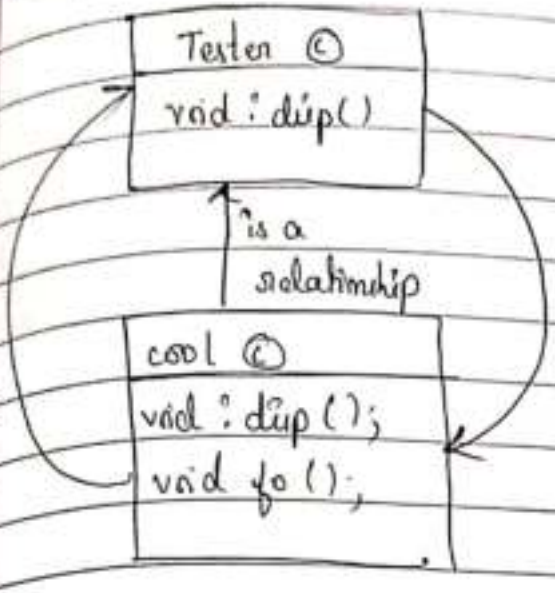
```
Sample S1 = new Sample()
SOP (S1.x);
```



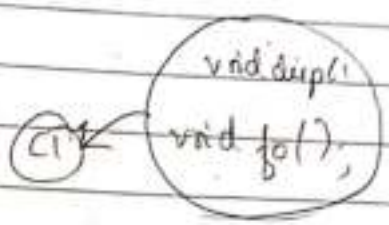

```
Demo d1 = new Demo()
SOP (d1.x);
d1.add();
```



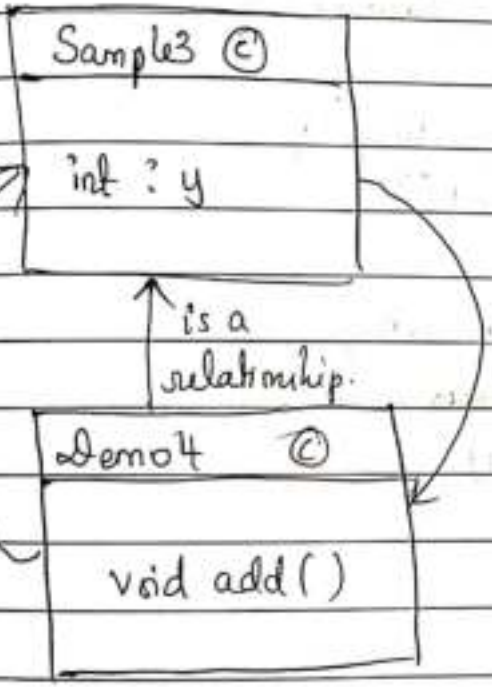
Example :-




```
Tester t1 = new Cool(); //upcasting.
t1.dúp();

Cool c1 = (cool)t1; //downcasting.
c1.dúp();
c1.fo();
```



Example :-




```
Sample3 s1 = new Demo4;
SOP(s1.y);

Demo4 d1 = (Demo4) s1;
d1.add();
SOP(d1.y);
```

In case a method over riding, even though it is upcasted we will get over ridden implementation

```
class Tester1
{
 int x=50;
}
class Demo1 extends Tester1
{
 void co()
 {
  S.O.P("cococooo coo");
 }
}
class Mainclass1
{
 public static void main(String[] args)
 {
  S.O.P("-------Upcasting");
   Tester1 t1 = new Demo1();
   S.O.P(t1.x);
   S.O.P("----Downcasting----");
      Demo1 d1 = (Demo1) t1;
       S.O.P(d1.x);
       d1.co();
 }
}
```

To achieve generation we use Inheritance

POLYMORPHISM

An object showing different behaviour at different stages of its lifecycle is called as POLYMORPHISM

Poly means "Many", Morphism means "forms"

In Polymorphism we have 2 types:
1) Compile time Polymorphism.
2) Run time Polymorphism.

COMPILE TIME POLYMORPHISM:

Since the method declaration getting binded to its definition at the compile time itself is called as "early binding".

Once the method declaration gets binded to its definition it cannot be rebinded, hence it is called as "static binding"

Method overloading is an example for "compile time polymorphism"

Def'n:
The method declaration getting binded to its definition at the compile time by the compiler based on the arguments passed is called "Compile Time Polymorphism"

Loose Coupling :-
If an enhancement is done internally it will not effect in usage.

RUNTIME POLYMORPHISM :-

The method declaration gets binded to its definition at the run time by the JVM based on the object created is called as 'run time polymorphism'

Since the method declaration gets binded to its definition, at the runtime, hence it is called as 'LATE BINDING'.

Once the method declaration is binded to its definition, it can be rebinded; hence it is called as 'Dynamic Binding'

Method overriding is an example for runtime polymorphism.

```
class Demo1
{
  void coo()
  {
    SOP("hahahaha");
  }
}

class Tester extends Demo1
{
  void cool()
  {
    SOP("Hahahaha");
  }
}

class Mainclass
{
```

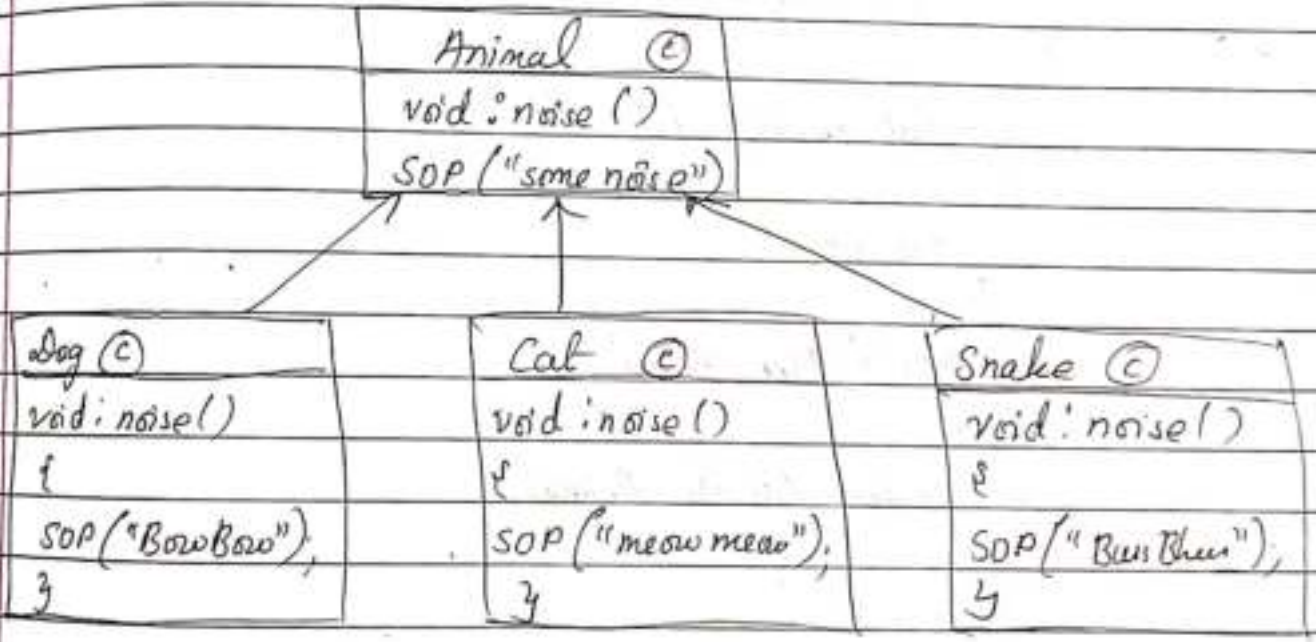
```
psvm (String[] args)
{
    Demo d1 = new Tester();
    d1.cool();
}
}
```

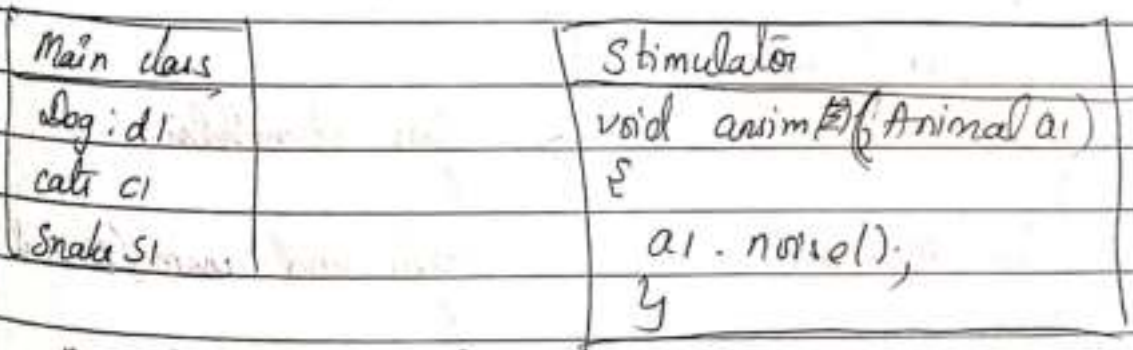
o/p: Hahahaha

```
void cool()
SOP("hahahaha");

void cool
{
SOP("hahahaha")
```

NOTE:-

In the case of method overriding even though it is upcasted we will get overridden implementation.





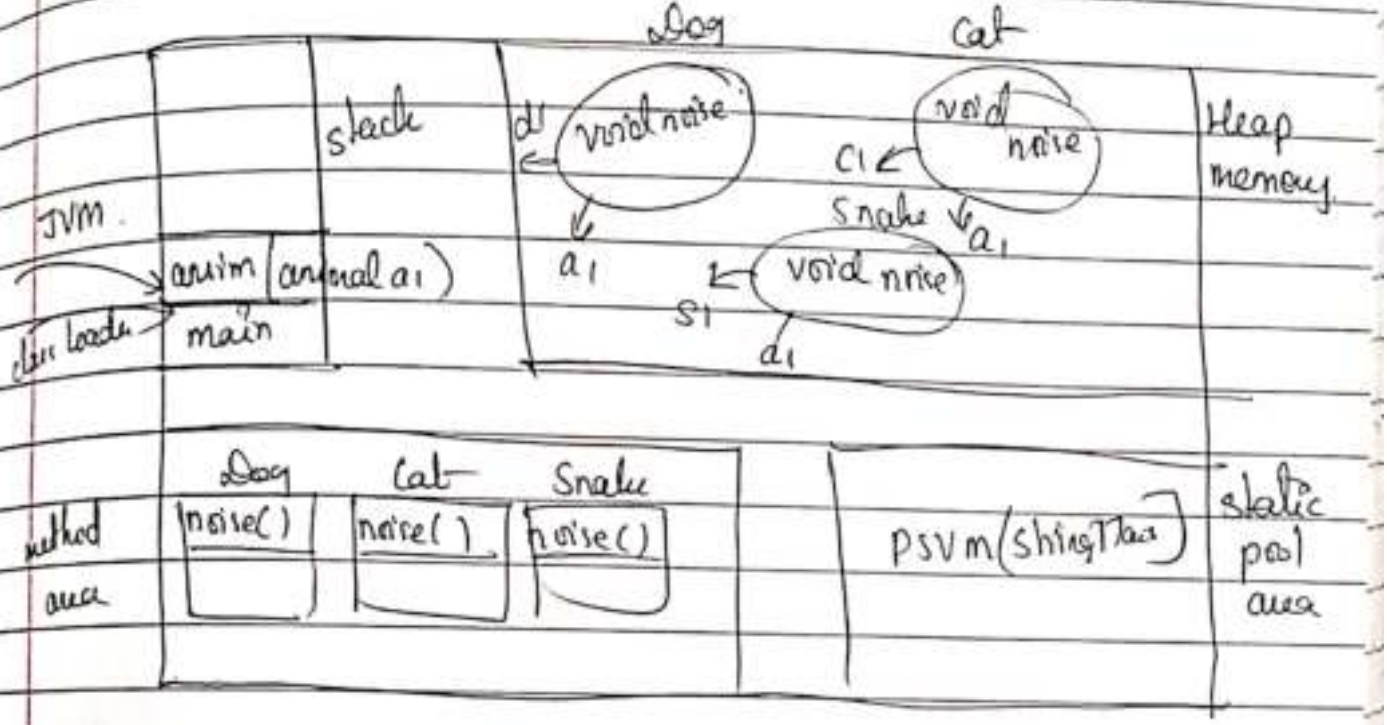

```
Animal a1 = new Dog()
    a1.noise();
Animal a2 = new cat()
    a2.noise();
Animal a3 = new Snake()
    a3.noise();
```

```
Pakage poly;
class Animal
{
 void noise()
 {
  SOP("some noise");
}}
class Dog extends Animal
 {
  void noise()
  {
   SOP("Bow bow..");
}}
class Cat extends Animal
 {
  void noise()
  {
   SOP("Meow Meow");
 }}
class Snake extends Animal
 {
  void noise()
  {
   SOP("Bus Bus");
 }
}
class Mainclass
{
 PSVM(String[] args)
{
 Dog d1 = new Dog();
 Cat c1 = new Cat();
```

→

```
class Stimulator
{
 static void anim(Animal a)
 {
  a1.noise();
 }
}
```

Snake s1 = new Snake();
Stimulator.ansim(d1);
Stimulator.ansim(c1);
Stimulator.ansim(s1);
}
}

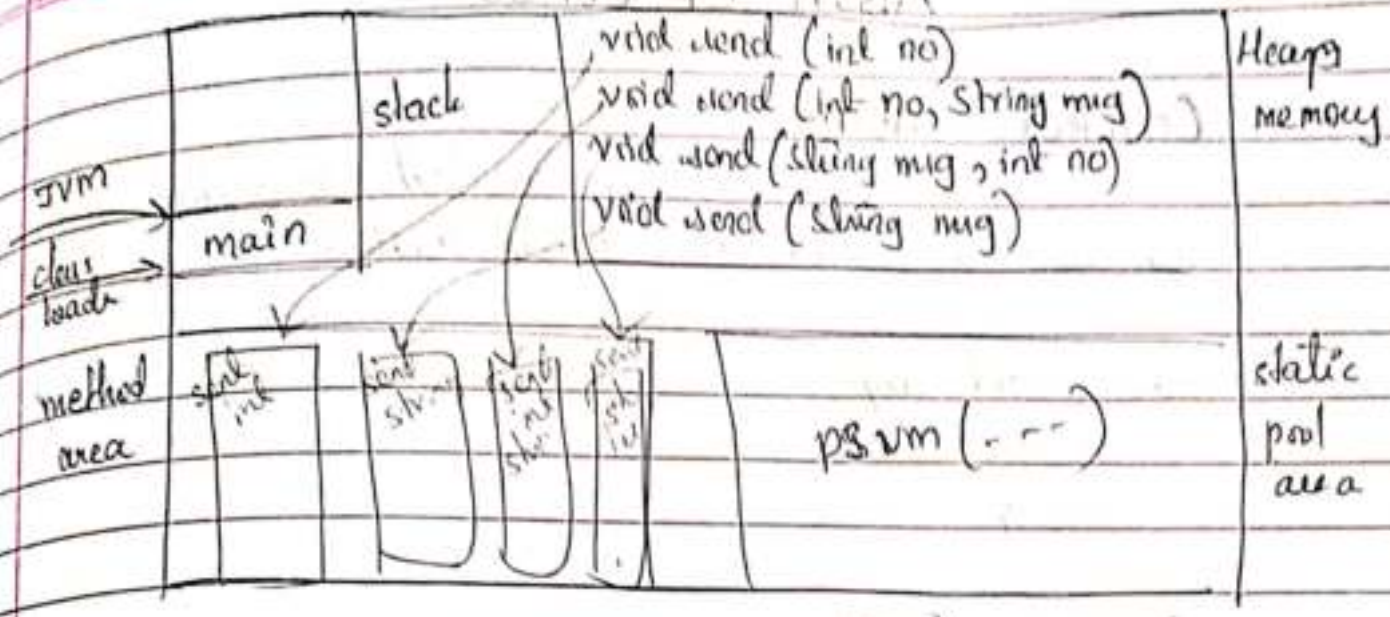


At the compile time it's going to check for method declaration & its difference

Compile Time Polymorphism Example :-

method overloading is an example for compile time polymorphism

```
class Whatsapp
{
void send (int no)
{
 S.O.P (no)
}
void send (int no, String msg)
{
 S.O.P(no+" "+msg);
}
void send(String msg, int no)
{
 S.O.P(msg+" "+no);
}
void send(String msg)
{
 S.O.P(msg);
}
Class Mainclass
{
public static void main (String[] args)
{
Whatsapp wa = new Whatsapp();
 wa.send(123);
 wa.send(true);
 wa.send("cool", 123);
 wa.send("Hi");
}
}
```



ABSTRACT CLASS

CONCRETE METHOD

Any method which has both declaration and defination is called as "Concrete Method"

Eg:
```
void disp()
{

}
```

CONCRETE CLASS

Any class which has only concrete methods is called as "Concrete Class"

Eg:
```
class Sample
{
    void disp()
    {
    }
}
```

ABSTRACT METHOD

Any method which is declared with the keyword abstract is called as "Abstract Method"

Eg: `abstract void disp();`

ABSTRACT CLASS

Any class which is declared with the keyword abstract is called as "Abstract Class".

Eg:
```
abstract class Demo1
{
}
```

→ If a class is having an abstract method then the class should be declared as abstract but wise-versa is not true

Eg:

```
abstract class Sample
{
abstract void disp();
}
```

→ An abstract class can have both concrete method and as well as abstract method.

Eg:

```
abstract class Demo
{
abstract void cool();
        void fool()
        {
        SOP("Hello");
        }
}
```

→ We cannot create object for abstract class and interface

→ The class which provides the implementation for the abstract methods, the subclass is also called as implementation class.

→ We cannot create abstract method as static, final, private.

→ If any one of the abstract method is not overridden in the subclass then the subclass should be declared as abstract.

→ Abstract class will have constructor.

```
abstract class Tester
{
abstract void dup();
abstract void cool();
}
class Demo2 extends Tester
{
  void dup()
{
  SOP("Hi");
}
  void cool()
  {
  SOP("Hello");
}
}
class Mainclass
{
public static void main(String[] args)
{
Demo2 d2 = new Demo2();
    d2.dup();
    d2.cool();
}
}
```

```
abstract class Demo2
{
abstract void test();
abstract void cool();
}
abstract class Tester2 extends Demo2
{
void test()
{
SOP("Hi");
}
abstract void cool();
}
class Sample2 extends Tester2
{
void cool()
{
SOP("Hello");
}
public static void main(String[] args)
{
Sample2 s2 = new Sample2();
s2.test();
s2.cool();
}
}
```

INTERFACE

- An interface is a java type.
- Interface is by default abstract and it is pure abstract body.
- In interface we have 2 members i.e.
  1. Variables
  2. Method.

- All the variables are by default static & final.
- All the methods are by default public and abstract
- We cannot create object class and interface.
- Interface does not support constructor.
- Java provides the keyword called implements to inherit the properties from interface to class
- The class which provides the implementation for the abstract method, the subclass is also called as implementation class
- If any one of the abstract method is not overridden in the subclass then the subclass should be declared as subclass.
- From an interface to interface, to inherit the properties we use the keyword extends.
- Each and every class extends object class, object class is the supermost class in the class type.
- Interface does not extends any class, interface itself is a supermost type.
- Through interface we can achieve 100% abstraction.

syntax:-

```
interface interface_name
{
    variable;
    method;
}
```

Ex:

```
(abstract) interface Sample
{
    int a=10;    // static final
    void disp(); // abstract public
}
```

```
abstract interface Tester
{
public abstract void disp();
public abstract void test();
}
class Sample implements Tester
{
public void disp();
{
  SOP("Hi");
}
public void test();
{
  SOP("Hello");
}
}
  class Main class
{
public static void main(String[] args)
{
```

```
Sample S1 = new Sample();
    S1.disp();
    S1.test();
```

(2)

```
abstract interface Sample1
{
public abstract void cool();
public abstract void test();
}
abstract class Demo2 implements Sample1
{
public void cool();
{
 SOP("I am cool");
}
} // public abstract void test();

class Tester2 extends Demo2
{
public void test();
 {
   SOP("Hi");
 }
 public static void main(String[] args)
 {
   Tester T2 = new Tester();
       T2.cool();
       T2.test();
 }
}
```

③
```
interface Nike
{
void shoe();
}
interface Puma extends Nike
{
void Bags();
void Shoes();
}
class Raj implements Puma
{
public void Bags()
{
SOP("jingilala bags");
}
public void Shoes()
{
SOP("jingilala Shoes");
}
public static void main(String[] args)
{
Raj R1 = new Raj();
R1.Bags();
R1.Shoes();
}
}
```

ABSTRACTION

Hiding the complexity of the system and exposing only the required functionality to the end user is called as Abstraction.

→ To achieve Abstraction, declare all the essential properties in the interface and provide the implementation in the sub class.

→ Create reference variable of interface type and initialize that reference variable with the implementation class object. This is how we can achieve Abstraction.

→ Through interface we can achieve 100% Abstraction

→ Through abstract class we can achieve upto 100% Abstraction.

NOTE :

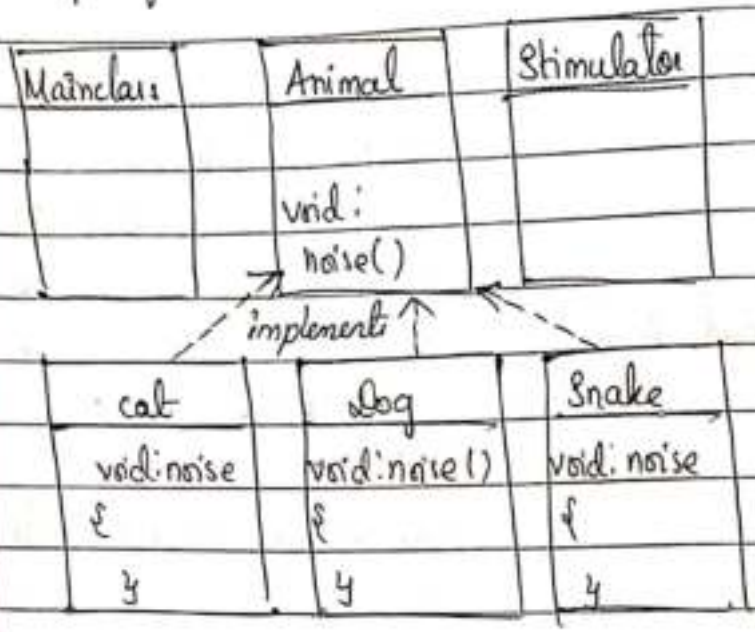
- When we don't know 100% implementation then we should go for interface.
  When we know partial implementation we go for abstract class.

The above points are to justify when to go for abstract class and when to go for interface

```
interface Audi
{
void wheels();
void break();
void engine();
}
class AudiA4 implements Audi
{
public void wheels();
{
SOP("Super wheels");
}
public void break();
{
SOP("Super break");
}
public void engine();
{
SOP("500cc engine");
}
class Mainclass
{
psvm(---)
{
AudiA4 A1 = new AudiA4();
A1.wheels();
A1.break();
A1.engine();
}
}
```

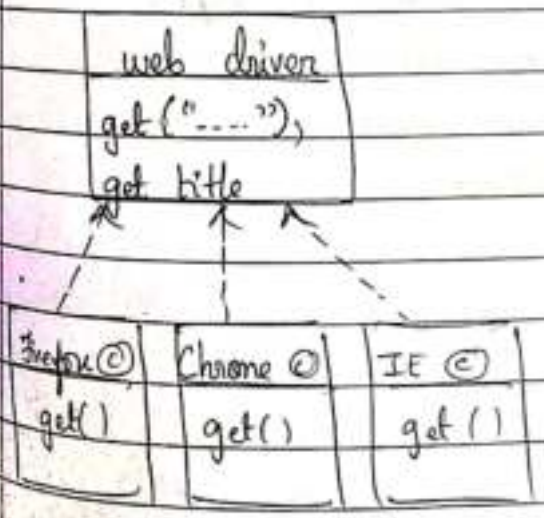
```
abstract class Audi
{
abstract void wheels();
abstract void break();
abstract void engine();
abstract void colour();
{
SOP("Black colour");
}}
class AudiA4 extends Audi
{
public void wheels();
{
SOP("Super wheels");
}
public void break();
{
SOP("Sudden Break");
}
public void engine();
{
SOP("500cc engine");
}
class Mainclass
{
psvm(-----)
{
AudiA4 A1 = new AudiA4();
A1.wheels();
A1.break();
A1.engine();
}
}
```

Example for Abstraction




```
interface Animal
{
void noise();
}
class cat implements Animal
{
public void noise()
{
SOP ("Meow Meow");
}}
class dog implements Animal
{
public void noise()
{
SOP ("Bow Bow");
}}
class snake implements Animal
{
public void noise()
{
```

```
SOP ("Buzz Buzz");
}}
class Stimulator
{
static void ansim (Animal a1) = new cat();
{
  a1.noise();
}}
class Mainclass
{
public static void main (String [] args)
{
 cat c1 = new cat();
 dog d1 = new dog();
 snake s1 = new ~~dog~~ snake();
 Stimulator.ansim (c1);
 Stimulator.ansim (d1);
 Stimulator.ansim (s1);
}
}
```




```
class Stimulator
{ static void ansim (Webdriver)
{ d1.get ("www.google.com");
}}
                    crome driver();
webdriver d1 = new ~~webdriver~~
  d1.get ("www.google.com");
class Mainclass
{
psvm (- - - -)
{
Stimulator.ansim (new firefox);
}
}
```

PACKAGE

It is a folder structure which is used to store similar kind of files. The package should be created always in the reverse order

Eg: www.gmail.com // → commercial

In any java file the 1st statement should be package statement. To import the files from 1 package to another package, java produces a keyword called 'import' where the files will be present virtually.

Import statement should be 2nd statement. We can have 'N' no of import statements file in a single java file. It can be any java type i.e. class, enum, interface and annotation.

```
① package Movies.englishActor
② import movies.kan movies.ActionMovies.KGF;
   import movies.kan movies.actionmovies.Jogi;
   import movies.kan movies.centi.Mungaru
③ class hybrid.Englukan movies
   {
   }
```

In eclipse when we develop multiple classes which ever class is public that class is eligible to have main method and it should be file name.

A.java

```
class C
{
}
public class A
{
p.s.v.m(...)
{
}
}
class B
{
}
```

ACCESS SPECIFIER

Access Speaifier is used to restrict the access from one class to another class (or) from one package to another package.

In java we have 4 Access Specifier.

01. Private only within the class
02. Default / Package level within the class & package
03. Protected within class, package, outside package with is subclass
04. Public.

Private.

Any member of the class declared with a keyword private is called as Private Access Specifier.

It can be accessed only within the class.

Default / Package level : specific

There is no ~~specifier~~ keyword specified for the access specifier. Then it is called as Default Access Specifier.

OR

If we declare any member of the class without any access specifier (or) keyword is called as Default Access Specifier.

Default members can be accessed

① Within the class
② Within the package.

Protected :

Any member of the class declared with the keyword protected is called as "Protected member of the class".

It can be accessed ① Within the class
② Within the package.
③ Outside the package with IS A RELATIONSHIP

Public :

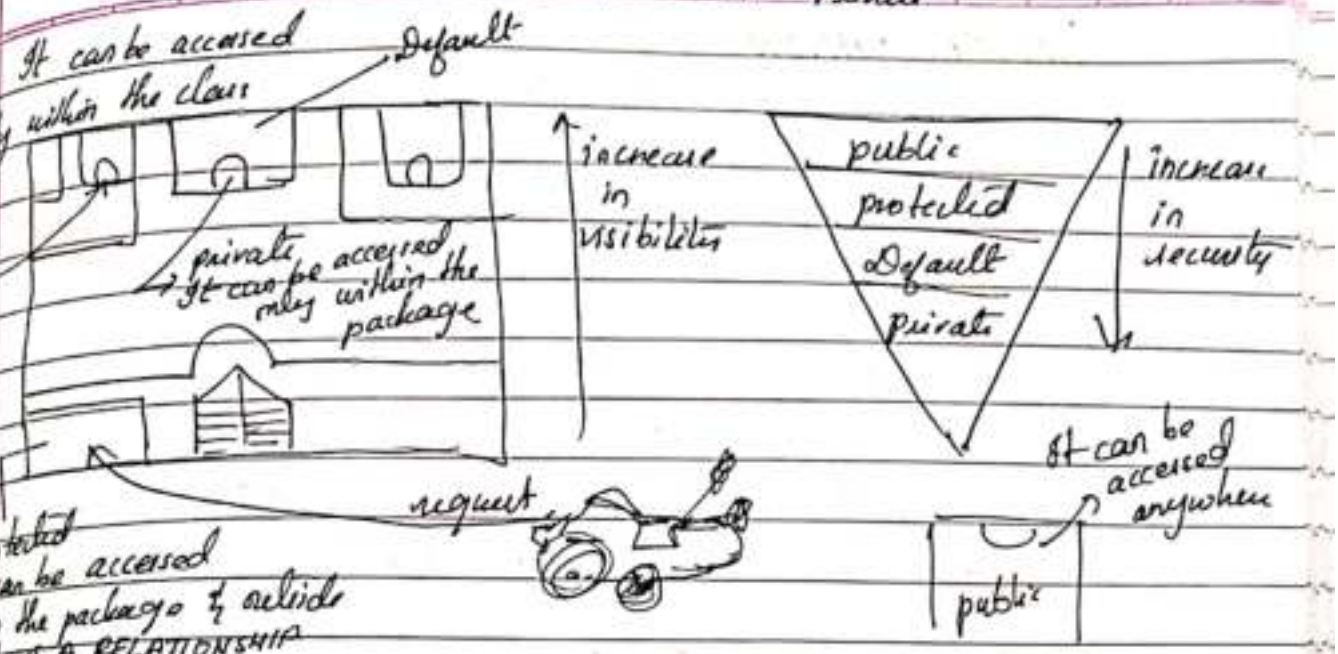
Any member of the class declared with a keyword public is called as "Public Access Specifier".

It can be accessed ① Within the class.
② Within the package
③ Outside the package (or) any where

NOTE : All the 3 members of the class can have all access specifiers and the class can have only 2 access specifier i.e. public and Default.

class Sample — public / Default

variable, Method, Constructor — public / Default / Protected / Private




```
package com.family.my family.
Public class Father
{ Private void atm()
{ Sysout("dad kaa atm");
} void car()
{ Sysout("dad ka Carrrr");
} Protected void Bike()
{
 Sysout("dad ka Bikeee");
} Public void cycle()
{ Sysout("dad ka cycleee")
}}
package com.family.my family
public class
{ P.S.V.m(——)
{
Father F1 = new Father();
  f1.car();
  f1.bike();
  f1.cycle();
}}.
```

create new Package
com.family.uncle family

```
package.com.family.uncle family
import com.family.my family.
                    father
Public class Aunty extends Father
{
P.S.V.m(——)
{
Aunty a1 = new Aunty();
  a1.bike();
  a1.cycle();
}}
```

encapsulation is used for protection. ex: Atm card pin; net banking pwd and provide security & provide access to particular person.

ENCAPSULATION

It is one of the oops principles. Java is by default encapsulated.

We cannot declare any variable outside the class.

We cannot have any print statement outside the class.

Declare the data member as private and restricting the direct access outside the class and provide the indirect access through public services called GETTERS AND SETTERS. It is called "Encapsulation"

What is java bean class?

Declare the data member as private and restrict the indirect access through public services is called as java bean class.

Declare all the variable as private

```
class Sample
{
private int a=10; → data hiding (we cannot access it directly)
                  ↗ we can use this only within the class
public int getA()
 {
  return a;
 }
 public void setA (int a)
 { this.a = a;
 }
  class Mainclass
 { p.s.v.m (———)
 { Sample s1 = new Sample ();
    int y = s1.getA ();
    SOP (y);
    s1.setA (123);
   SOP (s1.getA ()); // put directly in SOP ∴ getA() does not have void.
   SOP (s1.setA());
 }
}
```

```
① class ICICI
{
private int setpin=1234;
public int getatmpin()
{
return atmpin;
}
public void setA(int atmpin)
{ this.atmpin=atmpin;
}} class Mainclass
{ p.s.v.m(---)
{
ICICI card = new ICICI();
SOP(card.getatmpin());
card.setatmpin(1267)
SOP(card.getatmpin());
}
}
```

```
② class Facebook
{ private int pwd = 1234;
public int getpwd()
{ return pwd;
} public void setpwd(int pwd)
{
this.pwd = pwd;
}} public class Mainclass()
{
P.S.V.m(---)
{ Facebook F1 = new Facebook();
SOP(f1.getpwd());
f1.setpwd(4234);
SOP(f1.getpwd());
}
}
```

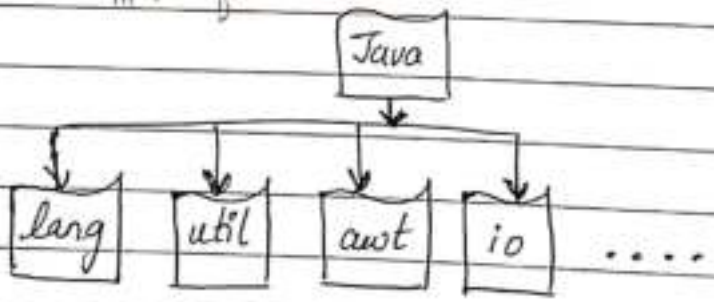
OBJECT CLASS

→ Object class is the super most class in java.
→ Each and every class extends Object class
→ Object class belongs to java.lang package, which will be imported by default.
In object class we have following methods as follows:

int hashcode()
boolean equals()
object clone() is used to take copy (it is not used)
String tostring()
void notify()
void notifyall()
void wait()
void finalize()

JAVA LIABRARY

Java Liabrary is nothing but inbuilt classes and interfaces. We have following libraries like in the folder structure.



ToString

ToString method is a non static, non final method of object class.

- It will be inherited to each and every class.
- Whenever we print the reference variable, toString method will be invoked implicitly, which will return fully qualified path in the form of string.
- Fully qualified path means package name.class name@hexadecimal number.

Example:

```
package poly;
public class Demo2 extends Object
{
public static void main (String[] args)
{
  Demo2 s1 = new Demo2();
  S.O.P (s1.toString());
}
}
```

→ generate fully qualified path

Here we not mentioning to write it will be virtually present

The signature of toString method is

```
public String toString
```

HASHCODE

Hash code method is a non static, non final method of object class.

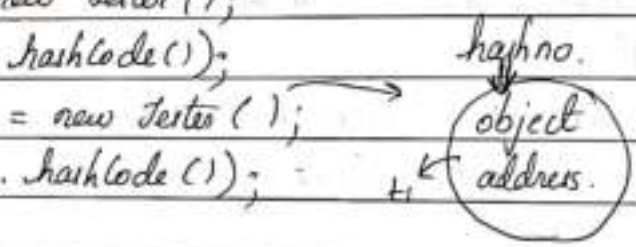
→ Hash code method will be inherited to each and every class.

→ Whenever we invoke the hashcode method, it will return the unique integer no. called hash no. based on the object address.

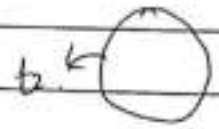
→ The hash code will be generated using hashing algorithm. The signature of hash code method is 'public int hashCode'

→ HashCode method shud be invoked explicitly.

① class Tester extends Object
{
public static void main (String [] args)
{
Tester t1 = new Tester ();
SOP (t1. hashCode());
Tester t2 = new Tester ();
SOP (t2. hashCode());
}
}



o/p : 12367134
56732193.



② package poly;
public class Tester extends Object
{
public int hashCode()
{
return 123;
}
p.s.v.m(—)
{

```
Tester t1 = new Tester();
  SOP (t1.hashCode());
}
}
```

EQUAL METHOD

Equal method is a non static, non final method of object class.

- → Equals method will be inherited to each and every class
- → Equals method is used to compare object address
- → The signature of equals method is "public boolean equals"

To compare the address we use equal method

Example:

```
package poly;
public class Tester extends Object
{
  p.s.v.m (—)
  {
    Tester t1 = new Tester();
    Tester t2 = new Tester();  →  Tester t2 = t1
    S.O.P(t1.equals(t2));           o/p true
```

o/p : False

The signature of equals method is
  public boolean equals

```
package [illegible];
public class Sample extends Object
{
  p.s.v.m (—)
  {
    Sample s1 = new Sample();
    Sample s2 = s1;
    SOP ( s1.equals(s2));
  }
}
```

STRING CLASS

String is a final class which belongs to java.lang package.

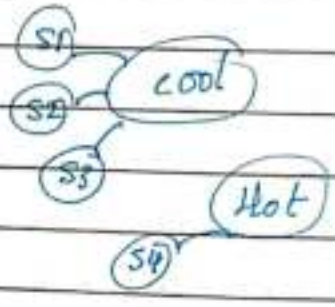
→ It should be imported to each and every package.
→ It is immutable [means it will not change the state of object]

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

Why it is immutable

When multiple reference variable are pointing to a single object and if one of the referance_variable de-reference with the current object it will not effect other reference variable. This is why it is immutable.

```
String s1 = "cool";
String s2 = "cool";
String s3 = "cool";
String s4 = "Hot";
```



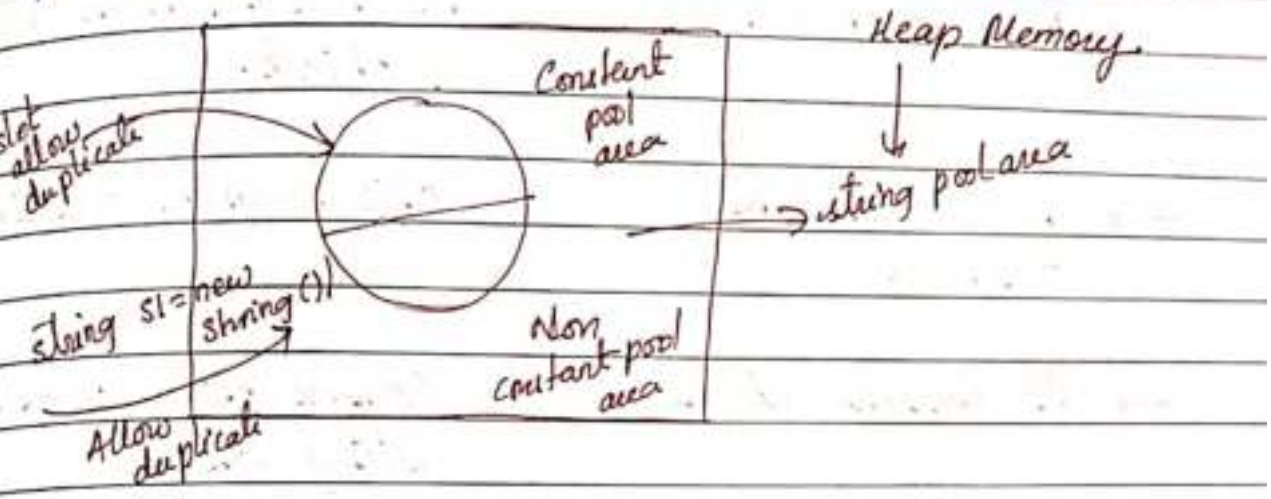
String object can be created in 2 ways
① String object with new operation.
② String object without new operations.

All the string object will be stored in string pool area which is under heap memory.

String pool area is divided into 2
01. Constant pool area
02. Non-constant pool area

→ All the string object which is created without new operator will be stored in constant pool area.

→ All the string object which is created with new operator will be stored in non-constant pool area.



→
```
package pdy;
public class Tester extends Object
{
  P.S.V.M ( ——— )
  {
    String S1 = new String ();
    String S2 = new String ();
    S.O.P ( S1 == S2)
    S.O.P ("climax" + S1. equals (S2));
    String S3 = "Hi";
    String S4 = "Hi";
    S.O.P ( S3 == S4);
  }
}
```

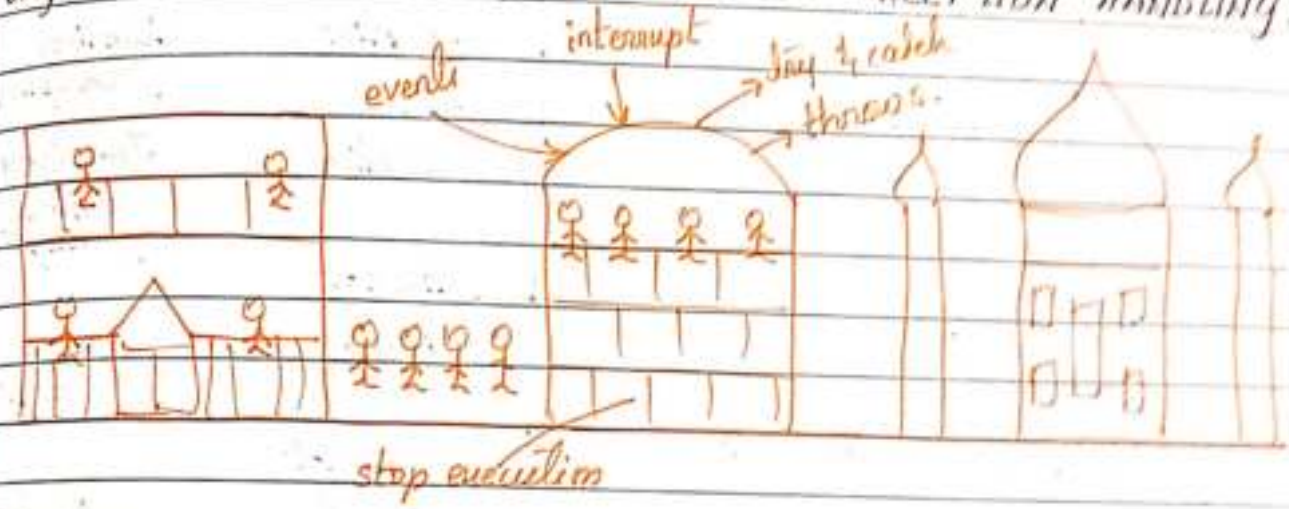
Difference between comparision operator and equal operator.

COMPARISION	EQUALS
→ It is an operator	It is a non-static method of object.
→ Operator cannot be overridden	Object can be overridden.
→ It compares address	Compare object values in string class.

Types of exception
→ checked except
→ unchecked except ← Runtime, ArrayIndex

EXCEPTION HANDLING
(for smoother execution)

It is an event triggered during the execution of the program which interrupts the execution and stops the execution abruptly / suddenly. Handling this event by using try and catch or throws is called as "EXCEPTION HANDLING".



Syntax:-

(1) try {
}
catch (Exceptionclass ref)
{
}

(2) try {
}
catch ()
{
}
catch ()
{
}

(3) try {
try {
}
catch ()
{
}
catch ()
{
}

(4) try {
}
finally {
}

(5) try {
}
catch ()
{
}
finally {
}

Throwable
Java.lang.Throwable

Error
Java.lang error
→ out of memory error
→ stack overflow error.

Exception
Java.lang exception.
(checked)
Compile time
IOE Exception
SQL Exception
File not found
Class not found except^n
Run time
Run time exception
Java.lang runtime exception
arithematic exception
Null pointer exception
Index out of bound
Illegal lang exception
:

ex:- try {
int = 1/0;
SOP ("cool");
} catch (Arithmeticexcept^n e) {

Once the exception occurs further statements of the try block will not get executed.

A reference variable can hold either object class or null.

THROW :
→ It is used to throw the exception of throwable type.
→ It will be written internally

* Once exception is addressed if we try to readdress then 'COMPILE TIME ERROR'

Example:

① Package exception topic
{
public class Sample1
{
p.s.v.m (——)
{
s.o.p("--- Main start ---");
try
{
int i = 1/0; // throw new Arithematic exception("/zero");
}
catch (Arithematic Exception e)
{ s.o.p("handled");
}
s.o.p("----- Main End -----");
}}

O/P :-
--- Main start ---
handled
--- Main end ---

② Public class Sample 2
{
p.s.v.m (——)
{
s.o.p("--- main start ---");
}
catch (Null pointer exception e) → reference variable [it can be any letter but in industry we use e]
{
s.o.p("handled");
}
s.o.p("---Main end---");
}}

O/P :-
---- Main start ----
handled
---- Main end ----

Between try and catch nothing should be written if not compile time error.

(3)

```
class Sample
{
  P.S.V.m(_____)
  {
    S.O.P("____Main Start____");
    int[] arr = {10,20,30,40};
    try {
      S.O.P(arr[8]);
    } catch(ArrayIndexOutOfBoundException e)
    {
      S.O.P("handled---");
    }
    SOP("____Main end____");
  }
}
```

NOTE:

For 1 try Block we can develop multiple catch blocks.

Example:

```
class Demo
{ P.S.V.M(_____)
  { S.O.P("_____Main start_____");
    try {
      int i = 1/0; // throw new ArithmeticException("/ by zero");
    } catch(ArrayIndexOutOfBoundException e)
    { S.OP("handled");
    } catch(ArithmaticException e)
      {
        S.O.P("Addressed");
      }
      S.O.P("_____Main end_____");
  }
}
```

O/P:-

Main start____
Addressed
____Main end____

- Compile time error occurs when we write something between try & catch block

- There may be multiple catch for 1 try block but there cannot be only one catch block will be executed.

- A reference variable can hold either null or object class.

Example:

```
class Demo
{
p.s.v.m (_____)
{
try
{ int i = 1/0; // Arithmetic Exception ------
  int[] arr = (10, 20, 30, 40);
  S.O.P (arr[8]); // Index Out Of Bound
}
catch (Array Index Out Of Bound Exception e)
{
S.O.P("handled");
}
catch (Arithmetic Exception e) <-------
{
S.O.P ("Addressed");
}}
```

O/P: Addressed.

Example:

```
class Sample
{
  p.s.v.m (_____)
{ S.O.P ("---- main starts ----");
try
{ int i = 1/0;
}
catch (Exception e)
{ S.O.P("handled");
}
catch (Arithmetic Exception e)
{ S'O P("Addressed");
}
}
```

O/P:

Compile Time error because 2 exceptional cases

Nested try catch

```
int [] arr = {10, 20};
try {
    S.O.P (arr[8]);
}
try {
    int i = 10/0;
}
catch (ArithmeticException e)
{
    S.O.P ("handled");
}
catch (Exception e);
{
    S.O.P ("caught");
}
}
```

→ throw can throw only 1 exception object at a time.

→ throw is used to create the except° & it is developed in method body.

FINALLY BLOCK :→

Finally Block is a block in Exceptional handling which will get executed mandatorly even though the exception is addressed or not addressed

We can develop finally block in 2 ways

① Try catch finally

② Try finally.

In Amazon / Re sale. Too many customers access the server and the server might get crashed even though the application crashes, the database closing connection statement should get executed, Hence we have to develop finally block.

Example:

```
class Sample
{
p.s.v.m (——)
{ s.o.p ("---- Main start ----");
try { int i = 1/0;
}
catch (ArithmeticException e)
{
s.o.p ("caught");
}
finally
{
s.o.p ("I am finally block");
}
s.o.p ("----- main end ----");
```

STACK UNWINDING :-)

If any exception occurs in any one of the method and if it is not addressed which will propagate the exception to the called method which in turn reaches main method and if main method propagates to JVM, hence JVM doesn't have any handler (throws / try & catch). The stack will get destroyed. Hence it is called as "STACK UNWINDING."

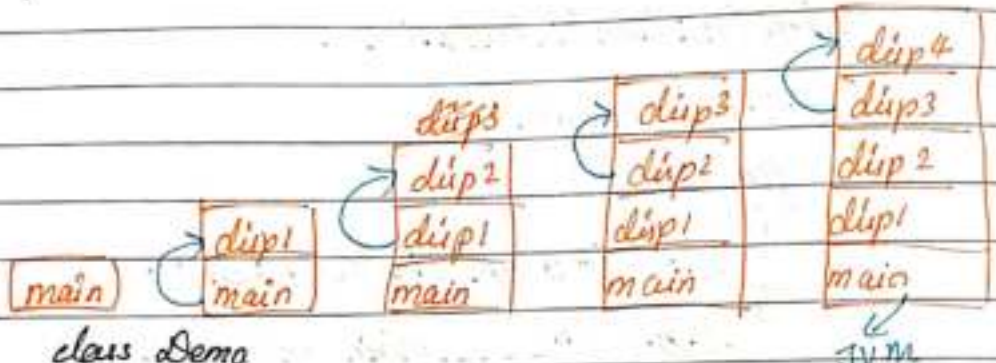
Finding the root of the cause is called as Ishikawa Technique.
or Fishbowl technique.
Print Stack Trace will help to find the root cause

PRINT STACK TRACE METHOD :-)

It is a non static method of throwable class which will be inherited to each and every class.

It will print the complete back trace of the stack where the error message has occured.

system.error [It will store] all the values & variables will be stored




```
class Demo
{
  static void disp4()
  { int i=1/0;
  }
  static void disp3()
  {
    disp4();
  }
  static void disp2()
  { disp3();
  }
  static void disp1()
  {
    disp2();
  }
  public static void main (String[] args)
  {
    S.O.P("---main start---");
  try {
    disp1(); // throw new ArithmeticException()
  }
  catch (ArithmeticException e) // = new ArithmeticExcept
  {
    e.printStackTrace();
  }
    S.O.P("-----main end-----");
}
```

WHAT IS EXCEPTION HANDLING ?

It is an event triggered during the execution of the program which interupts the execution and stops the execution abruptly. Handling this event by using try & catch or throws is called as "EXCEPTION HANDLING".

It is an abnormal statement which will terminate the program which we have to handle by using.

a. try and catch.
b. using throws.

All the abnormal statements should be developed in try block and address them in the catch block.

Exception are of 2 Types:

① Compile Time Exception
② Run Time Exception.

Compile Time Exception (Checked Exception)

Any exception which is caught by the compiler at the compile time, this is called COMPILE TIME EXCEPTION.

→ All the compile time exceptions can be addressed by using try and catch or throws.

→ Compile time Exception should be addressed at the compile time itself.

RUN TIME EXCEPTION (Unchecked)

Any Exception which is not caught at the compile time and found at the run time is called as RUN TIME EXCEPTION.

- Run Time Exception can be addressed by using try and catch or throws.

→ All the run time exceptions should be addressed at the run time itself.

① Once the exception occurs in the try block, further statement of the try block will not be executed.
② Once the exception is addressed, we cannot readdress if we try to do that then we get compile time error.
③ Exception class can address all the sub class exception which is either compile time or run time
④ For one try block there should be one mandatory catch.
⑤ We cannot develop any statement between try and catch block
⑥ For one try block we can have more than one catch block but only 1 catch block will get executed.

FINALLY BLOCK :-

Finally block is a block in exceptional handling which will get executed mandatorily even though the exception is addressed or not addressed.
We can develop finally block in 2 ways.
① Try catch finally
② Try finally.

STACK UNWINDING :-

If any exception occurs in any 1 of the method and if it is not addressed which will propagate the exception to the called method which in turn reaches main method and if main method propagated to JVM. Hence JVM does not have any handler (throws / try & catch) the stack will get destroyed. Hence it is called as STACK UNWINDING.

ERROR :-

Error is a class which belongs to java.lang package. Error is occured due to the system configuration.

DIFFERENCE BETWEEN ERROR AND EXCEPTION.

ERROR	EXCEPTION
01. Error is unpredictable	Exception is predictable.
02. Error is occured due to system configuration.	Exception is occured due to the mistakes done by the programmer.
03. Error cannot be handled	Exception can be handled.

DIFFERENCE BETWEEN THROW AND THROWS.

THROW	THROWS.
01. Throw is used to throw the instance of throwable type.	Throws is used to propagate the exception from 1 method to the called method.
02. Throw should be always developed in method body.	Throws should be developed in method declaration.

03. Throw is used to create the exception	Throws is used to propagate the exception.
04. Throw can throw only 1 object at a time	Throws can propagate more than 1 exception.

Example for Compile time Exception :—

```
class Shadidotcom
{
Static void Submit() throws ShadiException.
{
int age = 15
if (age >= 21)
{
S.O.P ("happy life");
}
else
{ throw new ShadiException("Invalid age");
}}
P.S.V.m (———)
{
try {
   Submit();
} catch (ShadiExcephion e)
{
   S.O.P (e.getmessage());
}}
class ShadiExcephion Extends Exception
{
   String msg;
   ShadiException (String msg)
{
   this.msg = msg
}
```

(3)

```
public String getmessage()
{
return msg();
}}
class FlipFlipException extends Exception
{
String msg;
FlipFlipException(String msg)
{
this.msg=msg;
} public String getmessage()
{ return msg;
}}
public class Flipkart
{
Static void Purchase() throws FlipFlipException
{ int Pant=4599;
if (Pant>=5000)
{ S.O.P("Great 200Rs discount");
} else {
throws new FlipFlipException("No Discount");
}}
P.S.V.m (←————)
{ S.O.P("——— main start ———");
try { Purchase();
} catch (FlipFlipException e)
{ S.O.P(e.getmessage());
} S.O.P("——— main end ———");
}
}
```

COLLECTION

→ Collection is an unified architecture which consists of CLASSES and INTERFACES.

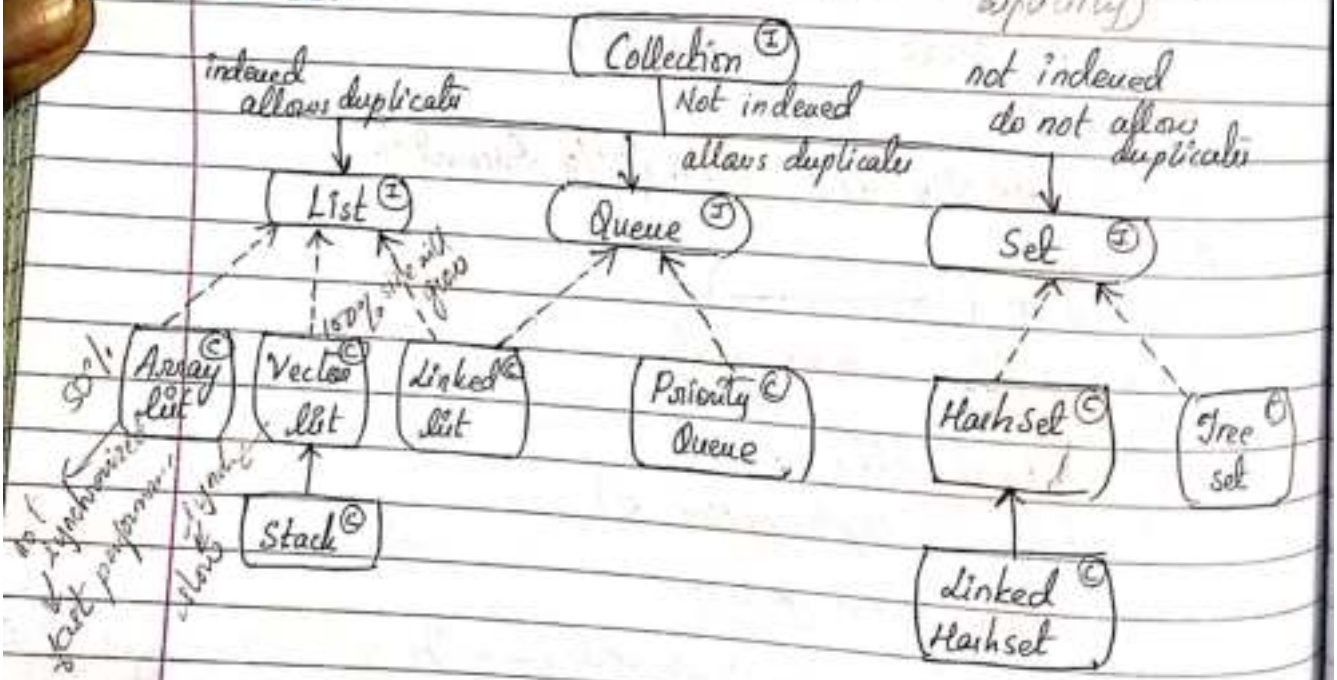
→ All the collection classes and interfaces belongs to Java.util package.

→ In order to overcome the drawback of array, we will go for collection

→ In array the size is fixed and we can store only homogenous type of data. But in collection the size is dynamic and we store heterogenous type of data.

Collection is an interface which has 3 sub interfaces :-

1. LIST.
2. QUEUE
3. SET.



```
package java.util
class Arraylist
{
  void add (object obj)
  void add (int index, object obj)
  void addAll (collection)
  void addAll (int index, collection)
  void remove ( )
  void removeAll ( )
  void retainAll ( )
  void get (int index)
}
```

Example: void add (object obj)

```
package collectiontopic;
import java.util.ArrayList
public class Sample1
{
  p.s.v.m (.——————)
  {
    ArrayList ll = new ArrayList();
      ll.add (10);
      ll.add (20.56);
      ll.add (10);
      ll.add ('A');
      S.O.P (ll);
  }
}
```

O/p :-

[10, 20.56, 10, A]

Example: void add (int index, object obj)

```
package collectiontopic;
import java.util.ArrayList;
public class Sample1
{
  P.S.V.M (———)
  {
    Arraylist l1 = new Arraylist();
        l1.add(10);
        l1.add(30);
        l1.add(10);
        l1.add('A');
        l1.add(1, "hello");
        SOP(l1);
```

O/P: [10, hello, 30, 10, A]

Example:

```
package collectiontopic;
import java.util.*;
public class Sample
{
p.s.v.m (———)
{
Arraylist l1 = new Arraylist();
        l1.add(10);
        l1.add(20);
        l1.add(30);
Arraylist l2 = new Arraylist();
        l2.add('A');
        l2.add('B');
        l2.add('C');
S.O.P ("------ b4 ------");
```

```
S.O.P ("l1 --> " + l1);
S.O.P ("l2 --> " + l2);
l1.addAll (l2);
S.O.P ("----- al -----");
S.O.P ("l1 --> " + l1);
S.O.P ("l2 --> " + l2);
}}
```

O/P :-

```
------ b4 ------
l1 --> [10, 20, 30]
l2 --> [A, B, C]
------ al ------
l1 --> [10, 20, 30, A, B, C]
l2 --> [A, B, C]
```

Example :

```
package collectiontopic
import java.util.*;
public class Sample
{
P.S.V.M ( ------- )
{
ArrayList l1 = new ArrayList();
        l1.add (10);
        l1.add (20);
        l1.add (30);
ArrayList l2 = new ArrayList();
        l2.add ('A');
        l2.add ('B');
        l2.add ('C');
S.O.P ("----- b4 -----");
S.O.P ("l1 --> " + l1);
S.O.P ("l2 --> " + l2);
    l1.addAll (1, l2);
```

```
S.O.P ("---- a4 ----");
S.O.P ("l1 --->" + l1);
S.O.P ("l2 --->" + l2);
}}
```

o/p: ------ b4 ------
l1 ---> [10, 20, 30]
l2 ---> [A, B, C]
------ a4 ------
l1 ---> [10, A, B, C, 20, 30]
l2 ---> [A, B, C]

REMOVE():

It will help to remove the elements from the collection object (it is used for string object).

Example:

```
package collectiontopic;
import java.util.ArrayList;
public class Sample1
{
  P.S.V.M (——)
{
ArrayList l1 = new ArrayList();
    l1.add ("goa");
    l1.add ("delhi");
    l1.add ("kochi");
    l1.add ("bangalore");
    l1.add ("mandya");
  S.O.P ("b4 --->" + l1);
   l1.remove ("delhi");
  S.O.P ("a4 --->" + l1);
    l1.remove (0);
  S.O.P ("a4 ---->" + l1);
}}
```

o/p:
b4 ---> [goa, delhi, kochi, bangalore, mandya]
a4 ---> [goa, kochi, bangalore, mandya]
a4 ---> [kochi, bangalore, mandya].

In collection the size is going to increase by 50% with the below formula

$$\text{capacity} = \text{current capacity} * \frac{3}{2} + 1$$

RETAINALL():

It will retain all the duplicates in l2 wrt with l2.

Example:

```
package collectiontopic;
import java.util.ArrayList;
public class Sample1
{
P.S.V.M(——)
{
ArrayList l1 = new ArrayList();
    l1.add(10);
    l1.add(20);
    l1.add(30);
    l1.add(40);
ArrayList l2 = new ArrayList();
    l2.add(30);
    l2.add(40);
    l2.add(50);
    l2.add(60);
S.O.P("------b4------");
S.O.P("l1-->"+l1);
S.O.P("l2--->"+l2);
 l1.retailAll(l2);
    S.O.P("-----a4-----");
    S.O.P("l1--->"+l1);
    S.O.P("l2--->"+l2);
}
}
```

O/P:-

```
------b4------
l1--->[10,20,30,40]
l2--->[30,40,50,60]
------a4------
l1--->[30,40]
l2--->[30,40,50,60]
```

REMOVE ALL ( ) :

It removes all the duplicate in l2 wrt l1.

Example :

```
public class Sample
{
ArrayList l1 = new ArrayList();
    l1.add (10);
    l1.add (20);
    l1.add (30);
    l1.add (40);
ArrayList l2 = new ArrayList();
    l2.add (30);
    l2.add (40);
    l2.add (50);
    l2.add (60);
  S.O.P ("----- b4 -----");
  S.O.P ("l1 --> "+ l1);
  S.O.P ("l2 --> "+ l2);
   l1.removeAll (l2);
  S.O.P ("------ a4 ------");
  S.O.P ("l1 --> "+ l1);
  S.O.P ("l2 --> "+ l2);
 } }
```

o/p :-

```
----- b4 -----
l1 --> [10, 20, 30, 40]
l2 --> [30, 40, 50, 60]
------ a4 -----
l1 --> [10, 20]
l2 --> [30, 40, 50, 60]
```

Object get (int index) :

Example:

```
class Sample
{
  p.s.v.m (_____)
{
Arraylist l1 = new Arraylist();
    l1.add (10);
    l1.add (20);
    l1.add (30);
    l1.add (40);
for (int i=0; i<l1.size(); i++)
  {
    s.o.p (l1.get(i));
}4
```

Array	Collection	String
arr.length	l1.size()	s1.length()

Collections :

It is a class which belongs to java.util.package. It has inbuilt methods like sort, search.

Example:

```
package collectiontopic;
impat java.util.Arraylist;
import java.util.Collections;
class Sample
{
  p.s.v.m (_____)
Arraylist l1 = new Arraylist();
    l1.add (10);
    l1.add (20);
    l1.add (30);
    l1.add (40);
```

Collections ©
sort()
binary search()

```
S.O.P("b4 ---> "+l1);
collections.sort(l1);
S.O.P("a4 ---> "+l1);
}}
```

O/P :
b4 ---> [10, 20, 30, 40]
a4 ---> [30, 40, 210, 410]

In Arraylist class the constructor is overloaded

1. Arraylist constructor without parameter [Arraylist()] this constructor will be invoked whenever an object is created which will initialize the default capacity as 10.
2. Arraylist (int initialCapacity) : it will initialise the capacity as per the user input.
3. Arraylist (Collection c) : It helps to copy from one collection to another collection object.

Example :-

```
package collectiontopics;
impot java.util.Arraylist;
import java.util.Collections;
class Sample
{
P.S.V.M(--)
{
Arraylist l1=new Arraylist();
l1.add(10);
l1.add(210);
l1.add(30);
l1.add(410);
Arraylist l2=new Arraylist(l1);
S.O.P("l1--->"+l1);
S.O.P("l2 --->"+l2);
}}
```

O/P :
l1 ---> [10, 210, 30, 410]
l2 ---> [10, 210, 30, 410]

ARRAYS :

Arrays is a class which belongs to java.util.package & it has inbuilt methods like sort, stream methods which helps to find the maximum value

* asList() : It is used to convert array to a list.

```
→ class Array
{
static List asList(int[] acc)
List l1 = new Arraylist();
l1.add(acc[0]);
l1.add(acc[1]);
l1.add(acc[2]);
return l1;
}
}
```

cont.....

Example :-

```
class Demo
{
p.s.v.m (——)
{
int [] abb = {10, 20, 30};
Array s. arlist (abb);
list l2 = as list ();
}
}
```

```
package collectiontopic;
import java. util. Arraylist;
import java. util. Arrays;
import java. util. Collections;
import java. util. list;
class Sample
{
p.s.v.m (——)
{
int[] arr = {100, 200, 310, 50};
list l2 = Arrays asList (arr);
}
}
```

```
class Demo Sample
{
static Sample dup ()
{
Sample x = new Sample ();
return x;
}
p.s.v.m (——)
```

```
Sample y = dup ();
int [] abb = {10, 20, 30};
int [] acc = abb;
}
}
```

default capacity of vector is 10

Vector:

Vector is a class which implement list interface.

Example:

```
1) package collectionhpic;
   import java.util.Vector;
   public class Sample1
   {
     p.s.v.m ( ——— )
     {
       Vector v1 = new Vector();
           v1.add (10)
           v1.add (20.56);
           v1.add (null);
           v1.add (10);
       SOP(v1);
       SOP("size ---> "+ v1.size());
       SOP("capacity ---> "+ v1.capacity());
     }
   }
```

O/P : [10, 20.56, null, 10]

size ---> 4

capacity ---> 10

Date: Yours

② package collectionspp;
import java.util.Vector
public class Sample1
{
p.s.v.m(——)
{
Vector l1 = new Vector(4);
l1.add(10);
l1.add(30.56)
l1.add(null);
l1.add(10);
l1.add('A');
SOP("size ——→"+ l1.size());
SOP("capacity--→"+ l1.capacity());
}
}

O/P : [10, 30.56, null, 10, 'A']
size----→ 5
capacity----→ 8

What is the difference between ArrayList and Vector

ArrayList	Vector
1. It increases its size by 50%	It increases its size by 100%
2. It is not synchronized	It is synchronized
3. Since it is not synchronized the performance is fast	Since it is synchronized the performance is slow.
4.	

Linked List

Linked list is a class which have dual property.

Example:

①
```
package collectiontopic;
import java.util.LinkedList;
public class Sample3
{
p.s.v.m( ---- )
{
LinkedList LI = new LinkedList();
        LI.add(10);
        LI.add(20.56);
        LI.add('A');
        LI.add("hello");
        LI.add(10);
      SOP(LI);
      SOP(LI.get(0));
      SOP("after get()---> "+LI);
      SOP(LI.poll.peek());
      SOP("after peek()---->"+LI);
      SOP(LI.poll());
      SOP("after poll()-->"+LI);
}
}
```

o/p:

[10, 20.56, A, hello, 10]

10

after get()---> [10, 20.56, A, hello, 10]

10

after peek()---> [10, 20.56, A, hello, 10]

10

after poll()---> [20.56, A, hello, 10]

Queue :-

Queue is the interface which extends collection interface.

It have 2 sub classes :

1. Linked List
2. Priority Queue.

Priority Queue.

Priority Queue is a class which implements Queue interface.

Example :

```
package collectiontopic;
import java.util.PriorityQueue;
public class Sample1
{
    P.S.V.M (------------)
    {
    PriorityQueue l1 = new PriorityQueue();
            l1.add(100);
            l1.add(30);
            l1.add(16);
            l1.add(10);
            l1.add(80);
        SOP("---------polluu-----");
        SOP(l1);
        SOP(l1.poll());
        SOP(l1);

        SOP(l1.poll());
        SOP(l1);
```

```
SOP(LI.poll());
SOP(LI);

SOP("------peekkkkkkk------");
SOP(LI.peek());
SOP(LI);

SOP(LI.peek());
SOP(LI);

SOP(LI.peek());
SOP(LI);
```

Opp:

```
------pollll----
[10, 16, 20, 100, 80]
10
[16, 80, 20, 100]
16
[20, 80, 100]
20
[80, 100]
------peekkk----
80
[80, 100]
80
[80, 100]
80
[80, 100]
80
[80, 100]
```

Set

Set is an interface which belong to java.util package.

It has 3 sub classes:

1. Hash Set
2. Linked Hash Set
3. Tree Set.

1. Hash Set.

Example:

```
package collectiontopic;
import java.util.HashSet;
public class Sample2.
{
 P.S.V.M (------)
 {
 HashSet LI = new HashSet ();
   LI.add (10);
   LI.add (30.56);
   LI.add ('A');
   LI.add (10);   SOP ("hello");
   add SOP (LI);
 }
}
```

O/P:

[A, 30.56, 10, hello]

Linked HashSet :

Example : package collection topic;

```
import java.util.LinkedHashSet;
public class Sample3
{
  P.S.V.M (————)
  {
  LinkedHashSet LI = new LinkedHashSet();
      LI.add (10);
      LI.add (20.56);
      LI.add ('A');
      LI.add ("hello");
      LI.add (10);
     SOP (LI);
  }
}
```

O/P :

[10, 20.56, A, hello]

TreeSet :

Example : public class Sample

```
{
  P.S.V.M (————)
  {
  TreeSet LI = new TreeSet();
      LI.add (100);
      LI.add (10);
      LI.add (16);
      LI.add (10);
     SOP (LI);
  }
}
```

O/P :

[10, 16, 100]

STACK

For-each loop

If we want to Traverse set which is not indexed, then we should go for "For-each-loop".

Syntax:-

```
for (array_type variable_name : array_name
                                (or)
                                collection reference_variable)
{
    ==========
}
```

Example:

(1)
```
String [] arr = {"Hi", "Hello", "Cool"};
for (String a : arr)
{
    SOP (a);
}
```

o/p: Hi
Hello
Cool

(2)
```
package collectiontopic;
import java.util.TreeSet;
public class Sample1
{
  P.S.V.m ( ---------- )
  {
    Treeset l1 = new Treeset();
        l1.add (100);
        l1.add (10);
        l1.add (160);
        l1.add (10);
```

```
if (l1.contains(160))
{
SOP("it contains 160");
}
else
{
SOP("it doesn't contain
}
for (object o1 : l1)
{
SOP (o1);
}
}
```

o/p :

it contains 160

10

100

160

What is Collection?

Collection is an unified architecture which consists of interface and class.

→ All the collections ~~belong~~ to classes and interface belong to Java.util Package.

→ In order to overcome the drawback of array we will go for collection. i.e.

① in collect^{n} the size is dynamic which it grows its size by 50%.

② The default capacity will be 10.

capacity = current capacity $* \frac{3}{2} + 1$

$= 10 * \frac{3}{2} + 1$

$= \underline{16}$.

③ It can store heterogenous type of data

→ In collection interface we have 3 sub interfaces

1. List
2. Queue
3. Set

LIST :-)

List is an interface which belongs to Java.util package

When you will go for List type of Collect^{n}?

Whenever we want to store upon index and allow duplicates the we have to go for List type of Collection.

Features of List

1. Size is dynamic
2. We can store heterogeneous type of data.

3. It is indexed type of collection.
4. It allows duplicates
5. It allows null.
6. It will follow order of insertion.
7. Since it is indexed type of collection we can do random access.
8. List interface have 4 sub classes i.e.
   (1) Array List
   (2) Vector List
   (3) Linked List
   (4) Stack.

→ ARRAY LIST :-)

Array list is a class which implements list interface.

Features of Array List:-

1.
2.
⋮

8. It will grow its size by 50%
9. It is not synchronised
10. Since it is not synchronised, the performance is fast.
11. 8

→ VECTOR LIST

Vector is a class which implements list interface

Features of Vector:

1
2
⋮

8. It increased its size by 100%.
9. It is synchronized

10. Since it is synchronized, the performance is slow.

→ LINKED LIST

Whenever we want proper order of insertion, then we should go for linked list.

Feature of Linked List :-

1
2
⋮

8. It inherits the properties from both list and queue.
9. It can use poll() and peek() method or get() method to fetch the data.

→ STACK

→ Stack is a class which belongs to java.util package.
→ Stack extends vector class.

It can perform the following operation like :-

1. push.
2. pop.
3. peek
4. search.

Feature of Stack :-

1.
2
⋮

8. It follows "last in first out"

QUEUE :-)

Queue is an interface which extends collection interface.
In queue interface we have 2 sub classes.

1. LINKED LIST
2. PRIORITY QUEUE

→ Priority Queue :-)

Priority Queue is a class which implements queue interface which is of pure queue.

Features of Priority Queue :-

1. Size is dynamic
2. We can store heterogenous type of data.
3. It is not indexed.
4. It allows duplicates.
5. It is auto sorted.
6. Since it is not indexed type of collection we cannot fetch the elements upon index.
7. In priority queue to fetch the elements we should use poll() and peek().

Poll() method

Poll() method is a non-static member of queue class, which helps to fetch the top most element of the queue and remove the element from the queue and reduce the size of the queue by 1.

Peek() method

It is a non-static method of priority queue class which helps to fetch the top most element of the queue and it will not reduce the size of the queue by 1.

SET :-)

Set is an interface which extends collection interface.

When do we go for set type of collection?

Whenever we want to store the elements not upon index and not allowing duplicates, then we should go for SET type of collection.

Inside interface we have 3 sub classes :-

1. Hash Set
2. Linked Hash Set
3. Tree Set.

→ Hash Set

Hash Set is a class which implements set interface

Features of Hash Set :-

1. Size is dynamic
2. We can store heterogenous type of data.
3. It is not indexed type of data.
4. It do not allow duplicates.
5. Since it is not indexed type of collection we cannot do random access.
6. It allows null.
7. It will not follow order of insertion.

→ Linked Hash Set

It is a class which extends hash set.

features of Linked Hash Set :-

1
2
:
:
7. It will follow order of insertion.

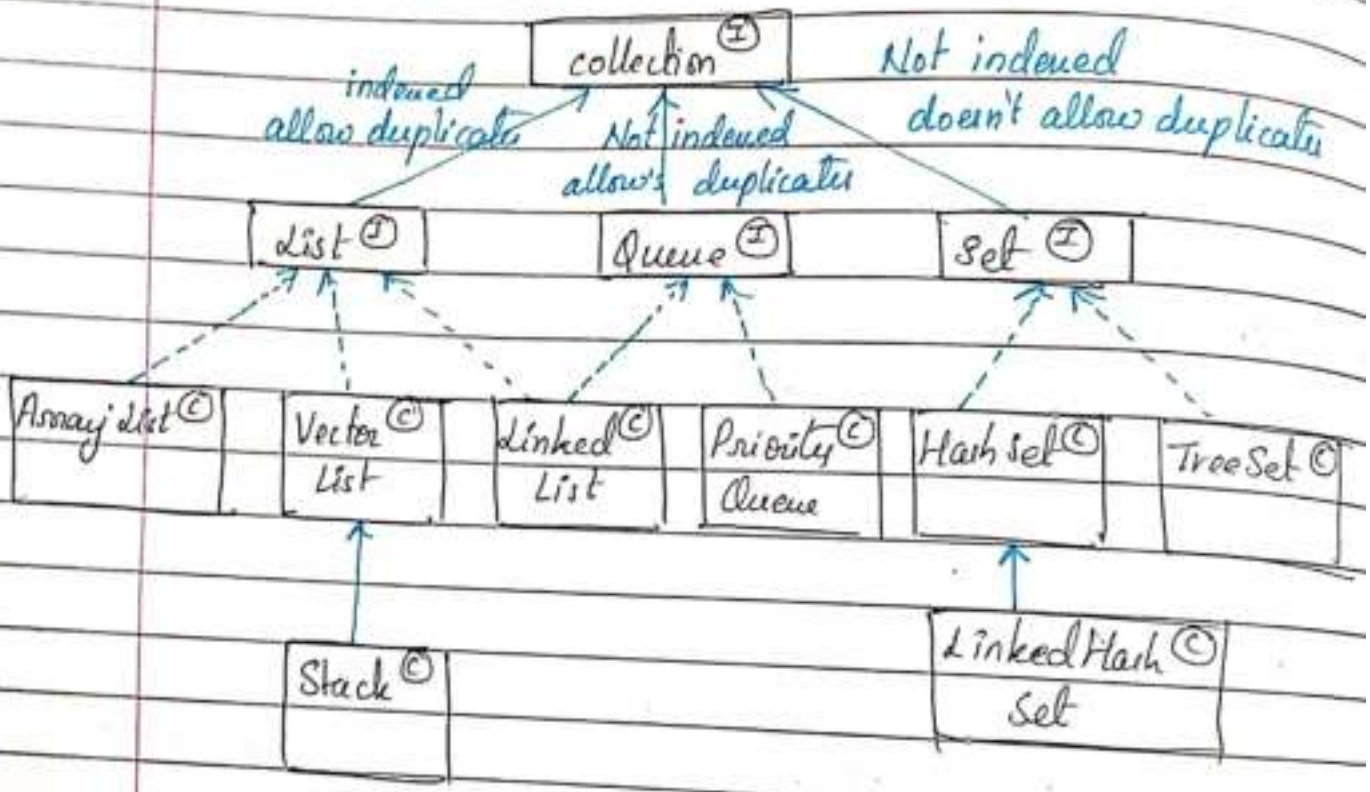
→ Tree Set.

Tree set is a class which implements set interface

Feature of Tree Set :-

1
2
⋮

7. It is completely auto sorted.



MAP

Map is an interface, which belongs to java.util package.

Whenever we want store upon key and value then we should go for 'MAP'.

Features of Map :-

1. Size is dynamic
2. It can store heterogenous type of data
3. It stores the elements upon key and value
4. It cannot have duplicate keys.
5. We can have duplicate values.
6. Map has 3 sub classes :-
   (1) Hash Map
   (2) Linked Hash Map
   (3) Tree Map.

→ Hash Map :-)

It is a class which implements map interface.

Features of Hash Map :-

1.
2.
⋮
6. It will not follow order of insertion.

→ Linked Hash Map :-)

Features of Haked Hash Map :-

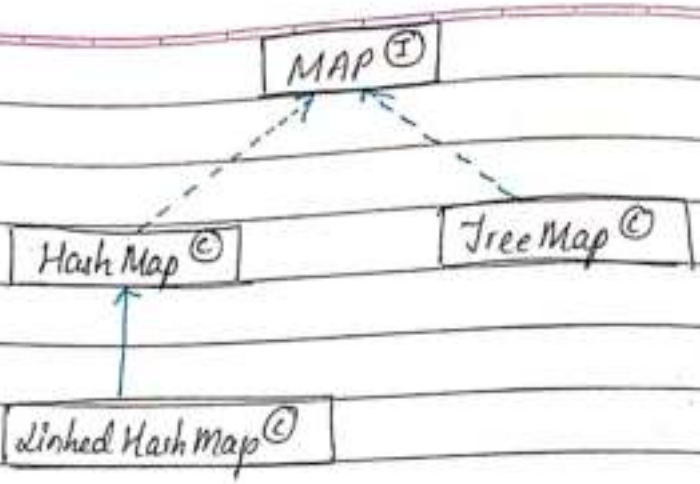
6. It will follow order of insertion.

→ Tree Map :-)

It is a class which implements Map interface

Features of Tree Map :-

6. It is completely auto sorted based on key.



Generic Class.

To achieve generic type we have to use angular braces <>
By using generics we can store homogeneous objects of a specified class type

Map is of generic type by default.

primitive datatype	Wrapper datatype
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean.

Example :-

(1)

```
package intro;
import java.util.HashMap;
public class Sample1
{
    P.S.V.M (_____)
{
HashMap<String, Integer> LI = new HashMap<String, Integer>();
```

```
LI.put("Ramya",127);
LI.put("Kaathik",127);
LI.put("Dad",125);
LI.put("Dad",123);
SOP(LI);
}
}
```

O/P:-

{Kaathik=127, Dad=123, Ramya=127}

(2)

```
package intro;
import java.util.LinkedHashMap;
public class Sample2
{
p.s.v.m(------)
}
LinkedHashMap<String, Integer> LI=new LinkedHashMap<String,Integer>();
LI.put("Ramya",127);
LI.put("Kaathik",127);
LI.put("Dad",125);
LI.put("Dad",123);
SOP(LI);
}
}
```

O/P:-

{Ramya=127, Kaathik=127, Dad=123};

(3) It sorts based on ascii values.

```
package intio
import java.util.TreeMap;
public class Samples
{
    p s.v.m (------------>)
{
TreeMap<String, Integer> LI = new TreeMap <String, Integer>();
        LI.put("Ramya", 127)
        LI.put("kaushik", 127)
        LI.put("Dad", 125)
        LI.put("Dad", 123)
    SOP(LI);
}
}
```

O/P:-

{Dad = 123, kaushik = 127, Ramya = 127}

CONSTRUCTOR OVERLOADING :

Developing multiple constructors within the class but variation in argument list is called Constructor overloading.

Variation in argument list means :

i) variation in the datatype.

ii) variation in the length of the arguments.

iii) variation in the order of occurance of arguments.

THIS CALLING STATEMENT :

It is used to call from one constructor to another constructor within the class.

RULES :

1. This calling statements should be first statement inside the constructor.
2. It is used only in the case of constructor overloading.
3. We cannot develop 2 this calling statements in a single constructor.

Example :

```
class Demo
{
Demo (int a)
{
  S.O.P (a);
}
Demo (String b)
{
  this (10);
  SOP (b);
}
Demo (int x, String y)
{
  this ("cool");
S.O.P (x + " " +y);
}
```

```
Demo (String y, int x)
{
this (10, "hello");
S.O.P(y+" "+x);
}}
class mainclass
{
P.S.V.M (——)
{
new Demo ("Hi", 126);
}}
```

O/P:-
10
cool
10 Hello
Hi 126.

SUPER CALLING STATEMENT

```
class Demo
{
Demo (String a)
{
S.O.P ("hahaha");
}}
class Tester extends Demo
{
Tester (double y)
{
Super ("Hi");
S.O.P("cool cool");
}}
class Sample extends Tester
{
Sample (int a)
{
Super (10.56);
S.O.P("hi");
}}
```

```
class Mainclass
{
p.s.v.m ( ——— )
new Sample (10);
}}
```

O/P:-
hahaha
cool cool
Hi
LO.

→ It is used to call from sub class constructor to the immediate super class constructor.

RULES:-

1. Super calling statement should be the first statement inside the constructor.
2. It is used only in the case of inheritance.
3. Super calling can be written implicitly or explicitly.

Example:

```
class Demo extends Object
{
  Demo ()
  {
  S.O.P ("cool");
  }}
  class Tester extends Demo
{
// default constructor will be created
}
class Sample extends Tester
{
  Sample ()
  {
  S.O.P("Hi");
}}
```

```
class Mainclass
{
p.s.v.m ( ——— )
{
new Sample ();
}
}
```

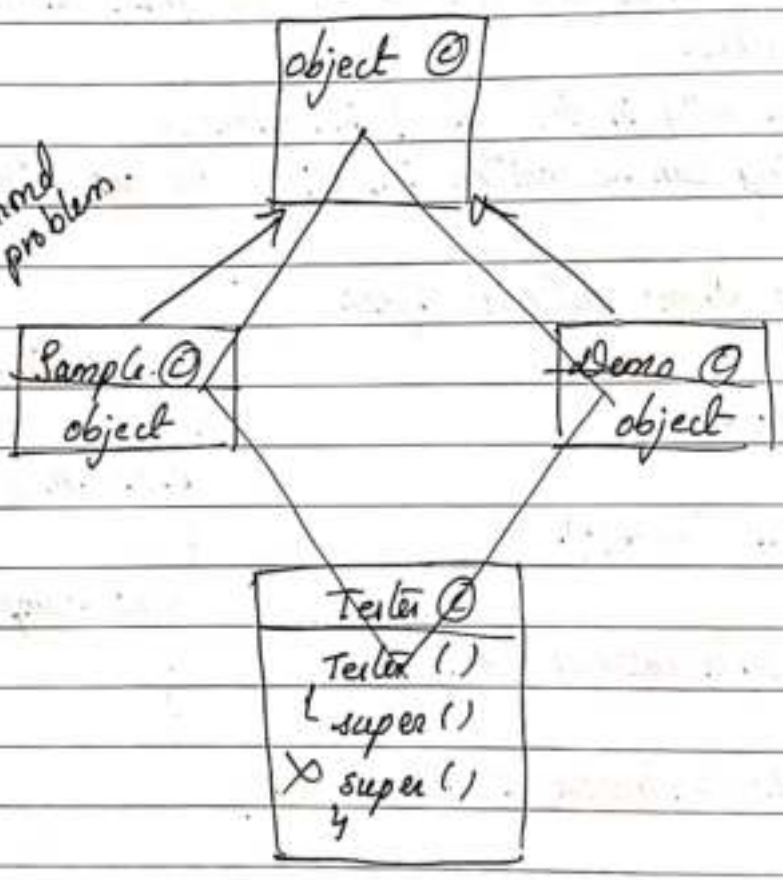
O/P:-
cool
Hi

INTERVIEW QUESTIONS:

1. Why multiple inheritance is not possible in Java through classes?

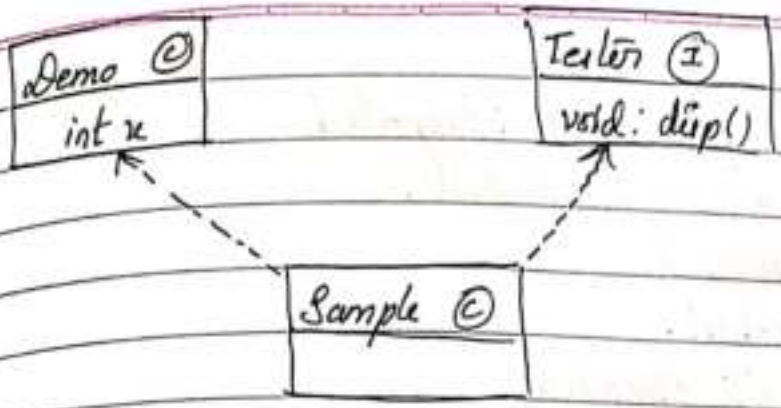
→ We cannot develop 2 super calling statements in the sub class constructor because super calling statement should be first inherited.

→ Due to diamond problem i.e. each and every class extends object class, the submost class will have ambiguity (confusion) from which super classes it shows be should be inherited.



→ We can achieve multiple inheritance through interface.

* interface does not support constructor. Hence we do not develop 2 super calling statements in the sub class constructor.
* interface does not extend any class, interface itself is a super most & hence there will be no diamond problem.




```
interface Demo
{
int x = 10;
}
interface Tester
{
void dup();
}
class Sample implements Demo, Tester
{
public void dup()
{
S.O.P("Hi");
}
P.S.V.M(————)
{
Sample s1 = new Sample();
        s1.dup();
        S.O.P(s1.x);
}}
```

2. What are Java features?

1. It is complied and interpreted.
2. It is platform independent.
3. It is secured.
4. It is robust.
5. It is multi threaded.
6. It is easy and simple to understand.

3. Why java is object oriented programming language?

Java is object oriented programming language because it supports the following concepts:

1. Encapsulation.
2. Polymorphism.
3. Inheritance.
4. Class & object.
5. Abstraction.

This is why it is object oriented.

4. What is the difference between this calling and super calling?

This calling	Super Calling.
1. it is used to call from one constructor to another constructor within the class.	it is used to call from one constructor to another constructor between the class.
2. it is used in the case of constructor overloading.	it is used in the case of inheritance.
3. it should be written explicitly	it can be written explicitly or implicitly.
4. it should be first statement inside the constructor	it should be first statement inside the constructor.

5. What is the difference between this keyword & super keyword.

This Keyword	Super Keyword
1. it is used to point to the current object	It is used in the method overriding.
2. when the local and global variables are same, to differentiate between them we use this keyword.	along with the overridden implementation if I need super class implementation then we use super method name.
3. it can be used only in the non static context	it can be used only in method overriding.

6. What is the difference between abstract class and interface.

Abstract class	Interface.
1. any class which is declared with the keyword abstract is called abstract class.	It is a java type which is by default abstract.
2. In abstract class we have 3 members.	in interface we have 2 members.
3. variables in abstract class can be static or non static	variables in interface are by default static & final.
4. method can have any access specifier.	the methods are by default public & abstract.
5. we can develop both concrete method and abstract method	it is by default abstract
6. abstract class will have constructor	it do not support constructor.

Constructor chaining:- subclass constructor calling its immediate super class constructor intern super class constructor calling its immediate super class constructor is called as constructor chaining.

7. Abstract class extends object class.	it is the supermost class which does not extend object class.
8. Through abstraction we can achieve constructor chaining	we cannot achieve constructor chaining.
9. We have to override all the methods in the sub class which is abstract.	by default all the methods should be overridden.
10.	

7. Why the main method is declared as Public static void main(String[] args) ?

→ Public is application level access where JVM should be able to access main method for execution.

→ Static is the modifier which helps to load the main method before execution.

→ The return type is void where main method does not return any value.

→ Main is the method name.

→ The parameters are of String[] (array) type because main method receives the input in the form of String[] array

```
int x = IntegerParseInt("123");
S.O.P(x)
```

→ converts from String to int.

5. What do you mean by System.out.println?

→ System is a class

→ out is a global static reference variable

→ println is a non-static method of printstream class.

Example:-

```
class Demo // PrintStream
{
    void dup() //println()
    {
    }
class Sample //system
{
// static PrintStream out = new PrintStream();
   static Demo d1 = new Demo();
   static int x = 10;

class main class
{
 P.S.V.M (_____)
{
 S.O.P (Sample x);
Sample d1.dup();
  S.O.P ("Hi");
}
}
```

WRAPPER CLASS:

Wrapper class are the inbuilt classes which belongs to java.lang package.

→ For each and every primitive data type we have the corresponding class and those classes are called as Wrapper class.

→ We can perform an operation called boxing and unboxing.

BOXING: Converting from primitive data type to ~~corresponding~~ wrapper class object is called as BOXING.

UNBOXING: Converting from wrapper class object back to its primitive data type is called as UNBOXING.

Boxing and Unboxing will happen implicitly.

Primitive Type	Wrapper class.
byte	byte
short	short
int	integer
long	long
double	double
char	character
boolean	boolean

Example :- Integer a1 = new Integer(10); // Boxing
SOP(a1);
int x = a1; // unboxing
SOP(x)

o/p
10.

```
Character c1 = new Character ('A');
        S.O.P (c1);

char ch = c1;                    O/P :-
    S.O.P (ch);                      A.
```

How the elements are stored in the form of object?

Whenever we add the elements, the primitive data type will get converted to the corresponding wrapper class object and it will be upcasted and stored in the form of object.

```
Example : List<Integer> l1 = new ArrayList<Integer>();
          l1.add (10);
          l1.add (20);                O/P :-
          l1.add (30);                    10
          S.O.P (l1);                     20
                                          30
```

```
Example : l1 add (10);
          Object obj = new Integer(10);
            S.O.P (obj);              O/P :-
                                          10.
```

WAP to store user defined objects in arraylist class object

```
package collectiontopic;
import java.util.ArrayList;
class Mobile
{
 void call()
 {
  S.O.P("missed call");
 }}
  class Sample
{
 P.S.V.m (_____)
{
Arraylist <mobile> l1 = new Arraylist <mobile> ();
      l1.add (new mobile());
      l1.add (new mobile());
      l1.add (new mobile());
Mobile o1 = (mobile) l1.get(0);
       o1.call();
}
}

class Mobile
{
int mob_cost;
String mob_name;
Sting mob_colour;
mobile (int mob_cost, String mob_name, String mob_colour)
{
this.mob_cost = mob_cost;
this.mob_name = mob_name;
this.mob_color = mob_color;
}
```

```
p.s.v.m ( ——— )
{
mobile m1 = new mobile (10000, "Redmi", "Red");
  s.o.p (m1. mob_cost);
  s.o.p (m1. mob_name);
  s.o.p (m1. mob_color);
}
}
```

REVERSE ORDER :

```
package collectiontopic
import java.util. Arraylist;
public class Sample 3
{
  p.s.v.m ( ——— )
  {
    Arraylist l1 = new Arraylist ();
        l1. add (80);
        l1. add (20.56);
        l1. add ('A');
        l1. add ("Hello");
        l1. add (10);
    for (int i = l1. size () -1 ; i >= 0; i = i - 2)
      {
        s.o.p (l1. get (i));
    }
  }
}
```

O/P :- 10 A 80

FILE HANDLING :

File is nothing but collection of similar kind of files or data.

In java file is a class which belongs to java.io.package which should be imported explicitly.

In file class we have non-static methods like

mkdirs() → make directories.

createnewfile() → it helps to create the file.

exists() → it checks whether the file is present or not, if it is present it will return true. else it will return false.

Delete() → which will help to delete the file in the given path.

```
package filehandlingtopic;
import java.io.file;
class Demo
{
 p.s.v.m (_____)
 {
  file f1 = new file ("D:\\marabatch");
  if (f1.mkdirs())
  {
   S.O.P ("folder created");
  }
  else
  {
   S.O.P ("not created");
  }
```

```
if (fl.exists())
{
S.O.P("folder exists");
}
else
{
S.O.P("folder doesn't exists...");
}
if (fl.delete())
{
S.O.P("folder is deleted");
}
else
{
S.O.P("folder is not deleted");
}
}
}
```

o/p:-
folder created
exists
file deleted

Write a pgm to create a text files.

```
package filehandlingtopic;
import java.io.file;
import java.io.IoException;
public class Sample1
{
P.S.V.M(_____) throws IOException
{
File fl = new File();
if (fl.createNewfile())
{
S.O.P("file created");
}
else
{
```

S.O.P ("file not created");
}
}

O/P :- file created.

→ File writter and File reader are the inbuilt classes which belongs to java.io package.

→ File writer class helps to write the data into the file.

→ File reader will help to read data from the file.

Write a pgm to write a data into the file.

```
package filehandling topic;
import java.io.IOException.
import java.io.File.
import java.io.FileWriter.
class Sample1
{
  P.S.V.M (————)
  {
  File f1 = new File ("D:\\athey.txt");
  FileWriter fw = new FileWriter (f1);
  fw.write ("hello i will get job");
  S.O.P ("data is written");
        fw.flush();
  }
}
```

O/P :- Data is written.

Write a pgm to read data from the file.

```
package filehandler;
import java.io.File;
import java.io.FileReader;
public class Sample
{
  p.s.v.m (———) // throws Exception
  {
  File fl = new File ("D://atthey.txt");
  FileReader fr = new FileReader (fl);
  char[] arr = new char [(int) fl.length()];
   fr.read (arr);
  String s1 = new String (arr);
      S.O.P (s1);
  }
}
```

O/P :- Hello i will get job

```
package filehandlingtopic;
import java.io.BufferedWriter;
import java.io.file;
import java.io.fileWriter;
import java.io.IOException;
public class Sample1
  {
    P.S.V.m (———)
}
  File fl = new File ("D://atthey.txt");
  FileWriter fw = new FileWriter (fl, true);
  BufferedWriter bfw = new BufferedWriter (fw);
      bw.write ("hello");
```

```
        bw.newLine();
        bw.write("hello");
        bw.newLine();
        S.O.P("data is written");
          bw.flush();
    }
}
```

```
package filehandlingtopic;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
public class Sample2
{
P.S.V.M(————) throws Exception
{
  File f1 = new File("D:\\Matthy.txt");
  FileReader fr = new FileReader(f1);
  BufferedReader br = new BufferedReader(fr);
    String s1 = br.readline();
      while (s1 != null)
      {
        S.O.P(s1);
          s1 = br.readline();
      }
  }
}
```

o/p :- Hello
Hello
Hello
Hello
Hello
Hello

* Date is a class which belongs to java.util.package which has inbuild methods which helps to get the data.

```
package filehandling topic;
import java.util.Date;
public class Sample3
{
  p.s.v.m (_____)
  {
  Date d1 = new Date();
  s.o.p (d1.getDate());        (todays date)
  s.o.p (d1.getMonth());       0-Jan  11-Dec
  }
}
```

o/p :- 17
11

* File input stream and File output stream are input class which belongs to java.io Package. It helps to read and write the media files.

WAP to read an image from the folder and write it back with a different name.

```
package qsp;
import java.io.file
import java.io.*
public class Sample1
{
  p.s.v.m (_____) throws Exception
  {
  File f1 = new File ("D:\\1.k.jpg");
  FileInputStream fin = new FileInputStream (f1);
```

```
byte [] arr = new byte [(int) f1.length()];
    fin.read(arr);

FileOutStream fout = new FileOutStream ("D:\\kabalidaa.
                                        jpg");

    fout.write(arr);
    fout.flush();
    S.O.P ("kabali is Back");
  }
}
```

Remove → f1.write(" ");

Write a pgm to achive singleton Design Pattern.

Singleton Design Pattern means restricting the Object creation by 1 through out the life cycle of the application is called as Singleton Design Pattern.

```
class Sample
{
  static int count = 0;
  int a = 10;
  static Sample s1;
  private Sample()
  {
    count++;
  }
  public static Sample getInstance()
  {
    if (count < 1)
    {
      s1 = new Sample();
    }
```

```
    return S1;
}
public static void setInstance(int y)
{
        S1.a = y;
}
class Mainclass
{
    P.S.V.M (_____)
    {
        Sample S2 = Sample.getInstance();
          S.O.P (S2.a);
        Sample S3 = Sample.getInstance();
          S.O.P (S3.a);
        Sample.setInstance(136);
          S.O.P (S3.a);
    }
}
```

An empty interface is called as marker interface.
Any interface which has only 1 abstract method is called functional interface.

Instance of Operator:

It checks does the object belong or instance belong to class hierarchy.

If it belongs then it returns true.

```
package qsp;
interface Animal
{
  void noise();
}
class Cat implements Animal
{
  public void noise()
  {
    S.O.P("meow meow");
  }
}
  class Demo12
  {
    P.S.V.M (------>)
    {
      Animal a1 = null;
      S.O.P(a1 instance of Animal);
}}}
```

o/p :- false

THREAD

Thread is an execution instance which owns its own CPU time and memory.

Thread is a class which belongs to java.lang package which has many inbuild methods like sleep methods.

* Thread pause the execution in miliseconds.

SLEEP METHOD :-

Sleep method is a static method which pause its execution for few miliseconds.

```
package threadtopic;
public class Sample1
{
 p.s.v.m (——) throws InterruptedExcepion
 {
  for (int i=1; i<=10; i++)
   {
     s.o.p(i);
     Thread.sleep (1000);
 }}}
```

o/p :-

MULTI THREADING :-

Processing multiple threads simultaneously is called as Multi Threading.

MULTI TASKING :-

It is a process of executing multiple task simultaneously is called as Multi Tasking.

SYNCRONIZED :-

Processing 1 by 1 thread is called as Syncronized. Processing multiple thread ~~by~~ simultaneously is called as "non-syncronized".

Write a pgm to non-synchronized multithreading.

```
package threadtopic
class Sample extends thread
{
  public void run()
  {
    for (int i=1; i<=10; i++)
    {
      S.O.P(i);
      try
      {
        Thread.sleep(1000);
      }
      catch (InterruptedException e)
      {
      }
    }
  }
class Demo implements Runnable
  {
    public void run()
    {
```

```
for (int i=100; i<=110; i++)
{
    s.o.p(i);
try
{
    Thread.sleep(2000);
}
    catch(InterruptedException e){
}
}
}
}
class Mainclass
{
  p.s.v.m(_____)
{
  Sample s1 = new Sample();
Thread th1 = new Thread(s1);
        th1.start();

Demo d1 = new Demo();
Thread th2 = new Thread(d1);
        th2 = start();
}
}
```

o/p:-

```
100
101
```

Scanned by CamScanner