Java - developed by James Gosling of Sunmicrosystem in (1995) released (1996)

Java is an object oriented platform independent programming language which is used to develop softwares.

Language - Language is a medium to communicate between two entities.

Programming language - it is a medium to communicate between human & a machine to develop softwares.

We have various programming languages like C, C++, Python & Java.

How a programming language is built -

english →	A-Z	→	words	→	Sentence	→	Paragraph	→	Story
Java →	A-Z 0-9 $, #	→	tokens	→	statement	→	Program	→	Software

Tokens

tokens - tokens are the smallest unit of program.

in tokens we have,

1. Identifier
2. Keywords
3. literals.
4. operators.

Scanned with OKEN Scanner

⑤ Seperators
⑥ comments

Identifiers- Identifier is a name given for the Java program

Keywords- keyword are the predefined Reserved words which has its own meaning.

In Java we have 50 keywords as follows-

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	Private	this
break	double	implements	Protected	throw
byte	else	import	Public	throws
case	enum***	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
**** added in 1.4
***** added in 5.0

Literals- Literals are the values which is used to perform some operations-

① numeric literals — Integers 9, 10, 11
— Decimal 9.1, 10.2, 10.4

string literals = "hello", "$", "ABC@$", "976" " "

Scanned with OKEN Scanner

③ character literal - 'A', 'B', 'C' 'd' '1' '+' ×

④ Boolean literal → true
→ false

Operators

operators are the symbols which is used to perform some operations on the operands.

operands
10 + 20
operators

UASRBLTA

operator type	category	precedence
unary	Postfix	expr++, expr--
	prefix	++expr, -expr, +expr, ~expr, !
Airthmatic	Multiplicative	*, /, %
	Additive	+, -
shift	shift	<<, >>, >>>
Relational	comparison	<, >, <=, >= instanceof
	equality	==, !=
Bitwise	bitwise And	&
	bitwise exor	^
	bitwise OR	|
Logical	Logical AND	&&
	Logical OR	||
Ternary	Ternary	?, :
Assignment	assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

Unary - It is called unary because it performs operation on one operand.

Unary plus - (if value is +) it gives +ve result | (if value is negative) then result is negative.

- It doesn't changes the sign.
- expression:= +a = +a.

int i = 1 | int i = -1
sop(+i); o/p = 1 | sop(+i); o/p = -1

unanary minus -
- It is used to convert positive value to negative value.

UNSPOKEN

example

① int i = 1; sop(-i); o/p = -1

② int i = -1; sop(-i); o/p = 1

Postfix - utlize then increment
Prefix - increment then utilize.

example - int a = 2;
int b = 3;
int c = 1;

a = a++ + b-- + --a + ++b + c++;
b = b-- + ++b + ++c + c++ + a--;
c = c++ + b++ + c-- + a-- + b++;

o/p = a=9, b=25, c=66

calc a = $\underset{2}{a}\overset{3}{++} + \underset{3}{b}\overset{2}{--} + \underset{2}{--a} + \underset{3}{++b} + \underset{1}{c}\overset{2}{++}$; a=11 b=3 c=1

b = $\underset{3}{b}\overset{2}{--} + \underset{3}{++b} + \underset{3}{++c} + \underset{3}{c}\overset{4}{++} + \underset{11}{a}\overset{10}{--}$ a=11 b=23 c=1

c = $\underset{4}{c}\overset{5}{++} + \underset{23}{b}\overset{24}{++} + \underset{5}{c}\overset{6}{--} + \underset{10}{a}\overset{9}{--} + \underset{24}{b}\overset{25}{++}$ a=10 b=24 c=

Scanned with OKEN Scanner

i=1
sop(i++)
o/p 1

but in an expression
i=2; j=2; k=2;
~~Sop(i)~~ k= i++ + j++ + k++
Sysout(i);
Sysout(j); o/p = i = 3
Sysout(k); j = 3
k = 6

i++ ya -- case me -
(expression me print hone par updated value print hoge)
ichain utilize binary increment wale hoge.
before

Q2 int a=1;
int b = 2;
int c = 1;

a= a++ + b-- + ++a + c++ + 2 + b--
c = b++ + ++c + b-- + a-- + 3 + b++ + 4
b = b++ + --c + b++ - a++ + a++ + (2×3)+4;

o/p a=11, c=20, b=39.

calc a= a++ + b-- + ++a + c++ + 2 + b--
1 2 3 1 2 1
a=10
b=0
c=2

c= b++ + ++c + b-- + a-- + 3 + b++ + 4
0 3 1 10 0
a=9
b=1
c=21

b= b++ + --c + b++ + a++ + a++ + (2×3)+4
1 20 2 9 10 +6 +4

a=11, b=39, c=20

Seperators-

seperators are used to seperate the given code
In seperators we have,
{} braces
[] brackets
() paranthesis
; semicolon
, comma

Scanned with OKEN Scanner

Comments - comments is used to provide additional information about the program, it dosen't gets executed.

Single line comments - //
block comments - /* */

Softwares required.

① editors - Notepad
edit plus.
② execution area - command prompt
③ 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	1.14
1.3	1.9	1.15
1.4	1.10	1.16
1.5/5.0	1.11	1.20

④ IDE (Integrated development enviornment)

eclipse → Standart edition
Netbeans → ME micro edition
→ EE enterprise edition.

⑤ Commands used in Java
cls - clear screen.
Javac - Java compiler
Java - Java interpretor
cd - change directory
cd.. - change to previous directory.

Scanned with OKEN Scanner

How to launch notepad -

Step1 - ON the System go to the desktop screen in the left bottom corner in the search box type notepad & double click on notepad. notepad will be launched.

How to launch command prompt.

① on the system go to the desktop screen left buttom corner type CMD & double click on the command prompt icon command prompt will be launched.

How to download JdK 1.8

Step1 Go to google type JdK 1.8 download

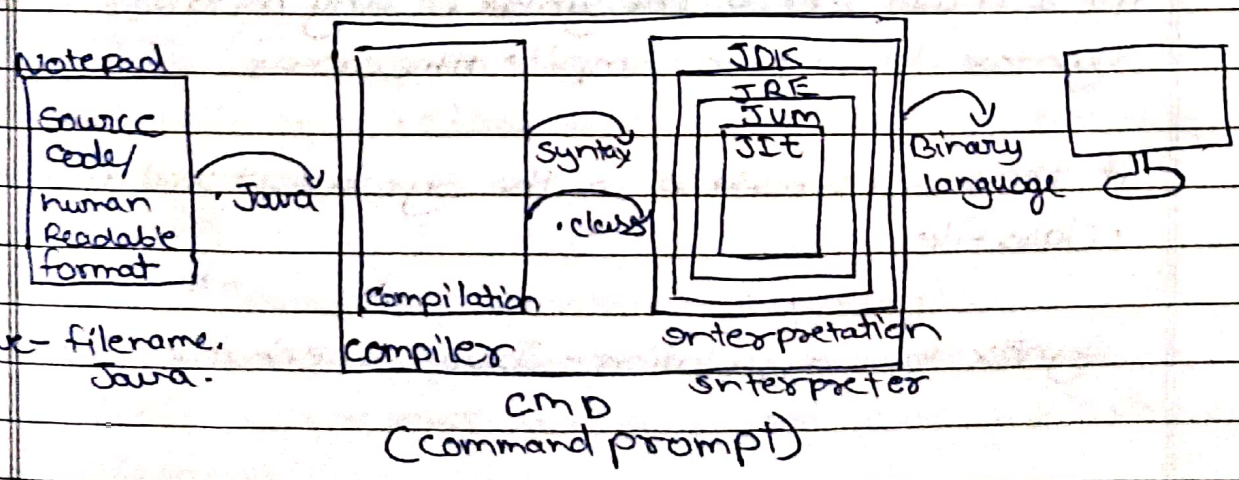
Step2 On the search results click on the first link which will take to the oracle website scroll little bit.

Step3 If your laptop is 32 bit download win X 86
If your laptop is 64 bit download win X 64.

How to check laptop is 32 bit or 64 bit

right click on this PC click on properties.

Java Architecture



Scanned with OKEN Scanner

Syntax -
for compilation - Javac filename.Java
for interpretation - Java. filename

compiler - check syntax
of mistake compile time error(CTE)
translate.

Interpreter - Read line by line
of mistake Run time error(RTE
translate.

As a developer we have to write the program in notepa
or editplus after writing we have to save it
with the extension of filename.Java.

After writing we have to perform two operation to
convert the from Java to binary language.
① compilation
② interpretation.

Compilation -
Compilations will be done by compiler. It will take .Java
file & it will check the syntax if any mitakes in the
syntax it will give compile time error.

if there is no mistake in the syntax it will generate
.class file

Syntax for compilation - Javac filename.Java.

Scanned with OKEN Scanner

Interpretation-

interpretation will be done by interpretor. It will take .class file it will read line by line if any mistakes it will give run time error. if there is no mistake it converts from classfile to binary language.

Syntax for interpretation- Java filename.

Note*- .Class file is also called as intermediate code which cannot be understandable by humans & computers.

types of interpreter components.

① JDK (Java development kit)
It will provide libraries to develop Java program.

② JRE (Java Runtime Enviornment)
JRE will provide enviornmental setup for the execution i.e memory for execution.

③ JVM (Java Virtual machine).
Jvm is a virtual machine which is not present physically & is whole responsible for execution.

④ JIT (Just in time)
It will take .class file & convert it in binary language.

Does Java is platform independent?
yes Java is platform independent because JRE is present in each and every electronic devices.

Scanned with OKEN Scanner

eb cLJ Shankar narayan.

papergrid

Date: / /

It works on WORA -

- Write once execute/Run Anywhere.

Java Types:-

In Java we have 4 types.

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

Class - Class is a blueprint or template to create object.

Write a program to print hello Java.

```
Class Sample          // class declaration
{                     // class body declaration
    Public static void main (String [ ] args) // main method Declaration
    {                 // main method body declaration
        System. Out. println ("Hello java"); // print Statement.
    }
}
```

identifier

```
:\users\faculty>cd Desktop
:\ users\faculty> Desktop>cd a26
:\ users\faculty> Desktop\a26>Javac Sample.Java
:\users\faculty> Desktop\a26>Java sample.
```

Scanned with OKEN Scanner

SDLC

Software development Life cycle it is a step by step or a standard procedure to develop the software application.

When SDLC will start -

whenever an user or company is ready to develop a software SDLC will start.

What will happen if SDLC is not followed.

- we will not get to know -

① how many engineers are required to develop a Software

② we will not get to know how many resources are required to develop the software.

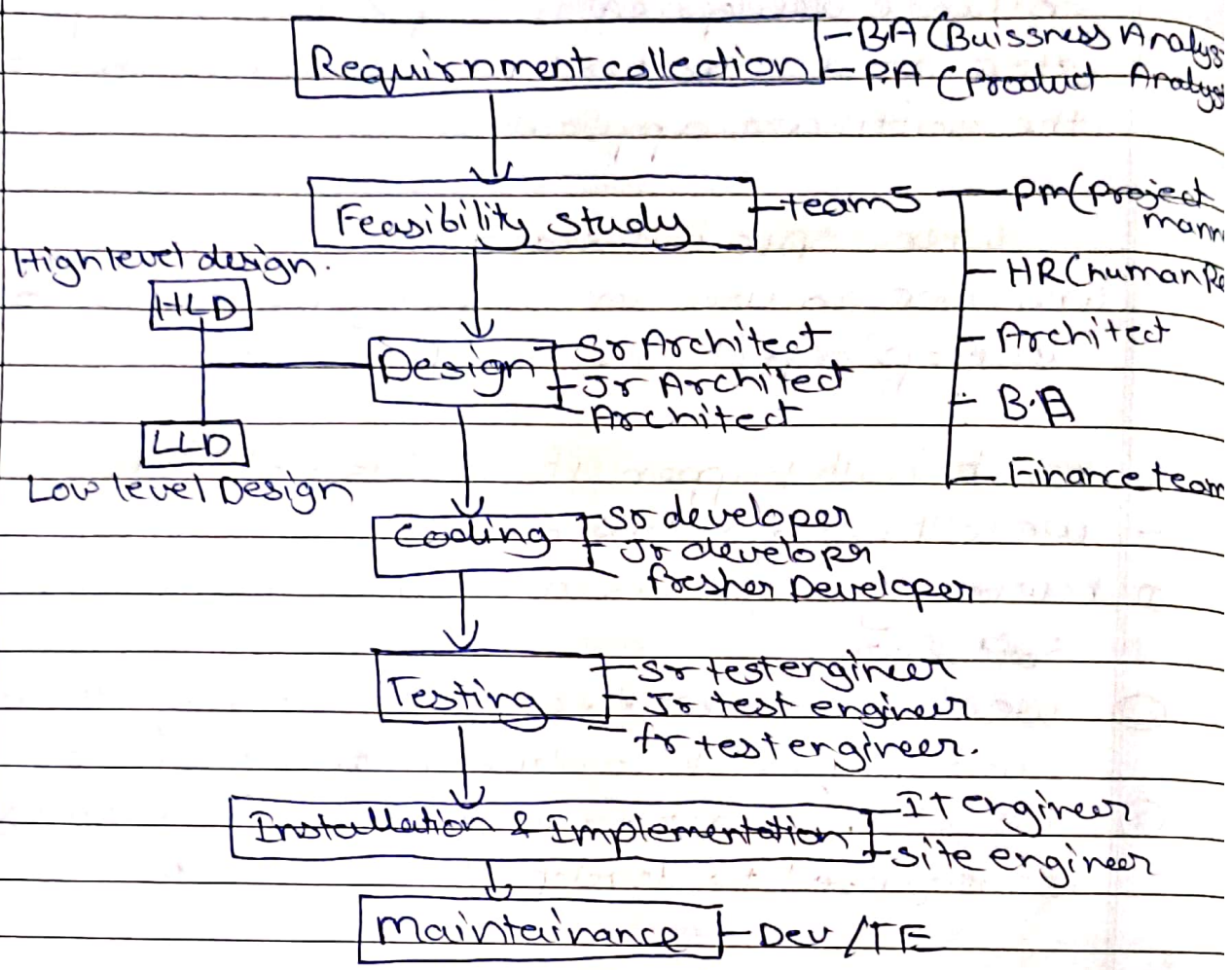
③ we will not get to know how much budget is required to develop the software.

Models in SDLC -

exp - ① waterfall model
② Spiral model
③ Prototype model
④ hybrid model
⑤ Agile model
⑥ V & V model (verification & validation model)

Scanned with OKEN Scanner

Stages of SDLC.



Requirnment collection - BA/PA.

Q Write a program to print student details.

```
Class Hello
{
Public static void main(str[] args)
{
    System.out.Println("prabhash");
    System.out.Println("Jspiders");
    System.out.println("Btech ECE");
}
}
```

Java executes from -

Top to bottom
Left to right

Anything concatinated with string is considered as a string.

```
Class Sample
{
Public Static void main (String[] args)
{
System.out.println("hello");
System.out.println(20+20);
System.out.println("the value is"+20+20);
System.out.println(20+20" is the value");
System.out.println("the value is" + (1/2));
System.out.println("the value is" + (10/20));
System.out.println("the value is" + (20+20));
}
}
```

O/p - hello
40
the value is 2020
40 is the value.
the value is 0
the value is 0.5
the value is 40.

```
Class Demo
{
Public static void main (String[] args)
{
System.out.println (20.5);
System.out.println("hello"+(2.5/2));
```

Scanned with OKEN Scanner

```
System.out.println(2/2)
System.out.println(1/2.0);
    }
}
```

O/P = 20.5
hello 1.25
1
0.5

Variables

Variable is a name of memory location which is used to store some value or data & it can be changed n no of times during execution.

Variable are of two type:

1. Primitive type
2. Non primitive type or class type or Reference type

① Variable declaration.

Syntax - datatype Variablename;
int a;

② Variable initialization

Syntax Variable name = value;
a = 10

③ Variable declaration & Initialization in a Single line.

int a = 10;

④ Variable utilization

System.out.println(a);

⑤ Variable copy from 1 to 2

Scanned with OKEN Scanner

```
int a=10;
int b=a;
sop(a);
sop(b);
```

(6) Variable Re intialization-

```
int x=90;
x=80;
sop(x);
x=70;
sop(x);
```

what is the difference between primitive data type.

The difference between primitive data type are size.

byte → 1byte → 8bits
short → 2byte → 16bits.

-2^{n-1} to $2^{n-1}-1$ range.
-2^{8-1} to $2^{8-1}-1$
byte = -128 to 127
short = -32768 to 32767

Type	size in bits	Range
byte	8	-128 to 127
short	16	-32768 to 32767
int	32	-2^{31} to $-2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	$1.4e^{-045}$ to $34e+038$
double	64	$4.9e^{-324}$ to $18e+308$
char	16	0 to 65,535
boolean	1	true or false.

Scanned with OKEN Scanner

Real time examples of variables

Q Gopal Joined Jspiders by paying the fees 40000 his section is A section. Write a program to display gopal details.

```
Class Student
{
Public static void main (String[] args)
{
    String stdname = "Gopal";
    int fees = 40000;
    char Sec = 'A';
S.O.P ("Student name is" + Stdname);
sop ("student fee is" + fees);
Sop ("Student see is" + sec);
}
}
```

Q car name, car price, car color;

```
class Car
{
Public static void main (String[] args);
{
    String name = "BMW";
    int price = "5,00,000;
    String color = "Red";
    Sop(name);
    Sop(price);
    Sop(color);
}
}.
```

Scanned with OKEN Scanner

Q write 10 programs with real time example on variables.

① mobile cost
mobile modelname.
mobile color

② carcost
car modelname
car type (petrol/dsly)

③ schoolname
school_grade
school_strength

④ tucost
tubrand
tu_type

⑤ bike_cost
bike_brand
bike_color

⑥ course_name
university_name
examcost

⑦ laptopname
laptopcost
laptop brand

⑧ homename
home cost
home color.

⑨ emaIl_id
emaIl_sal
employee_grade

⑩ cloth_color
cloth cost
clothsize (L, XL, M)

Q Write a program to demonstrate logical operators

in logical operators we have logical and & logical OR.

Relational operator-

```
Class student
{
    PSVM (String[] args)
    {
        System.out.println(10>2);
            SOP (5<10);
            Sop (8>=9);
            Sop (5<=5);
            Sop (8!=7);
            SOP (10==10);
    }
```

Scanned with OKEN Scanner

O/p - true.
true
false
true
true
true.

And

A	B	O/P
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	1

exor

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	0

exnor

A	B	O/P
0	0	1
0	1	0
1	0	0
1	1	1

Q WAP to demonstrate logical and

```
class student
{
P sum(string[] args)
 {
  SOP((10>2)&&(8>4));
  SOP((5<10)&&(8<5));
  SOP((8>=9)&&(5>3));
  SOP((5>=5)&&(3!=2));
  SOP((8!=7)&&(5>9));
 }
}.
```

O/P true
false
False.
true
false.

Q WAP to demonstrate logical OR.

```
Class student.
{
P sum (string [] Args)
{
  S.O.P ((10>2) || (8>4));
  S.O.P ((5<10) || (8<5));
  S.O.P ((8>=9) || (5>3));
  S.O.P ((5>=5) || (31=2));
  S.O.P ((81=7) || (5>9));
}
}.
```

O/P true
true
true
true
true

O/P is true or false.

Logical operators work on logical condition & Bitwise operators work on bits.

Q. WAP to demonstrate Bitwise operator?

Scanned with OKEN Scanner

8 4 2 1	0101	1000	0100
~~0000~~ 0111	1000	0011	0011
0010	1101	1011	0000
0010			

papergrid

Date: / /

Q

```
Class Student
{
Public static void main (String[] args)
{
  Sop(5/8);
  Sop(8/2);
  Sop(3 & 4);
  Sop(7 & 2);
  Sop( );
}
}.
```

O/P - 13
11
0
2

Q WAP to demonstrate ex-OR

```
Class Student
{
Public static void main (String[] args)
{
  Sop (4^3);
  Sop (8^2);
  Sop (7^3);
}
}
```

8 4 2 1
0 1 0 0
0 0 1 1
1 0 0 0
0 0 1 0
0 1 1 1
0 0 1 1

O/P - 7
10
4

Shift operators = It is used to perform shift operation on binary values.

In shift operators we have,

① Left shift <<
② right shift >>
③ unsigned right shift >>>

① Left shift - it will shift the bit to left side.

ex - ① int a = 10   8421
1010
a<<2;   101000
32168421 = 40
sop(a)
o/p 40

② int a = -10   o/p -40   same for (-ve) left shift.
a<<2;
sop(a)

Shortcut = a>>6
$2^6 \times a$ = left shit result.

② right shift = It will shift the bit to ~~left~~ Right side.

ex - int a=10;   8421 x x   Shift two bits right or delete last 2 bits
a>>2;   1 0 ~~1 0~~
sop(a)   0010

o/p = 2

for negative no - remember take 2's compliment then do right shift.
for two's compliment -
- Take 1's complement (do 0 as 1 & 1 & 0)
exp   a = 5   8421
0101
1's comp 1010

- 2's compliment add 1 in end.

Scanned with OKEN Scanner

1010 now 0+1=1
+1 1+1 = 010 carry
1011 = (-5) 1+0=1
0+0=0

int i = -5
i >> 2;
Sop(i)

8421
1011
.0010 = -2

o/p = -2

shortcut works only on +ve
even = n/2 for 1 shifting
odd = (n-1)/2 for 1 shifting only int

5 >> 3
o/p = 0

8421
0101

$\frac{5-1}{2} = \frac{4}{2} = 2$ ① $\frac{2}{2} = 1$ ② $\frac{1}{2} = 0.5$ 0 Ans

shifting 1 time left gives multipule by 2 &
shifting 1 time right gives factor by 2

③ unsigned right shift = for positive it works exactly like right shift

for negative. it will replace the signed bit by zero.

int i = -7
~~7777~~, i >>> 2;
sop(i)

- convert to 32 bit binary -

-7 take 2's compliment

1 - - - 111111111001

8421
0111
-7 S1000
+1
1001

shift right side by 2 & add 0 on left most side. (if 2 bits are shifted then 2 zero's to be added (or)

00 - - 1 - - - - - - - - 11110

result = 1073741822

Ternary operator = It is a conditional operator.

syntax -

variable = (condition) ? expression if true : expression if false;

exp int a = 10;
int b = 20;
~~int c = (a > b) ? sop(yes) : sop(no)~~
int c = (a > b) ? yes : no;
sop (c)

o/p = NO

storing in variable is not necessary you can directly use print statement in expression.

Variable's classification

Variables can be classified into local & global.

① **Local variable** -

- Any variable declared within a method is called as a local variable
- the scope of local variable is from the begning of the method to the end of the method.
- It cannot be classified into static & non static.
- It will not have default values.

Scanned with OKEN Scanner

⑤ Local variables should be initialized before utilization.

exp- class Sample
{
Public static void main (String[] Args)
{
int a;
a = 10
sop(a);
}
}

Global variables-

1) Any variable which is declared outside the method inside the class is called as global variable
2) the scope of the global variable is from the begning of the class till the end of the class.
3) It will have default values.
4) It can be classified into static & non static
5) once the global variable is declared immediately in the next line we cannot initialize or reinialize else it will show CTE.

exp- Class sample
{
Static int x=50;
Public static void main(String[] args)
{
sop (x)
}
}

Scanned with OKEN Scanner

papergrid
Date / /

datatype	defaultvalue
byte	0
short	0
int	0
long	0
float	0.0f
double	0.0d
char	'\u0000' (unicode)
boolean	false
String	null

Q WAP to display pizza prize, pizzasize, flavours.

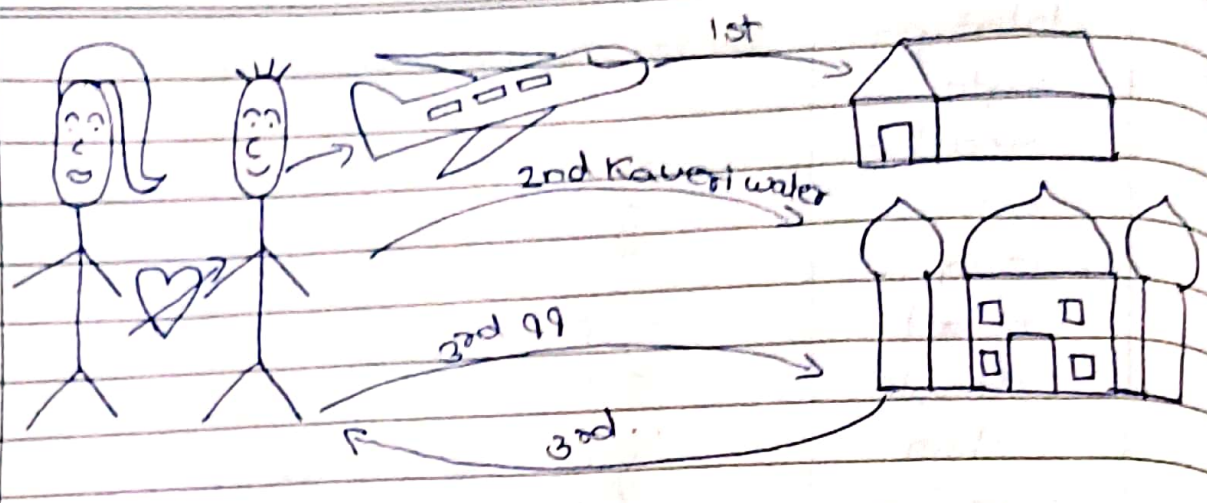
```
class pizza
{
static int pizzasize=7;
static int pizzaprize=540;
static string pizza flavor="paneer";
public static void main (String[] args)
{
    SOP (Pizza size);
    sop (pizza prize);
    SOP (pizza flavour);
}
}
```

HW Q write 10 programs with realtime examples with global variable.

Method/function member

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

Scanned with OKEN Scanner



Syntax -

Accessspecifier	modifier	returntype	method(argument) name	optional
-Private -default/ Package - Protected Public	used to restrict access	static NonStatic	void int double float String class	identifier camelcase - lower to upper addNumber

```
a   Classsample
    {
default- static void add(){
            int a=10;
            int b=20;
            int c= a+b;
            sop(c);
          return;
        }
        static void sub()
            int x=40;
            int y= 30;
            int z= x-y;
            Sop(z);
        }.
```

Scanned with OKEN Scanner

```
psvm(str[]args)
{
  sop("main starts");
   add();
   sub();
   add();
   sop("main ends");
  }
}
```

method convention -

1) method should start with lower case
2) 2nd word should be in upper case
3) No spaces allowed.
4) can start with Alpha, $, _

Q WAP to multiply 3 No's using method.

```
class multiply
{
  Static void mul()
  {
     int a = 2;
     int b = 3;
     int c = 7;
    sop(a * b * c);
   return;
  }
  psvm(str[]args)
  {
    sop("multiple is");
    mul();
  }
}
```

O/P - multiple is 42

Q WAP to add 3 no's using methods-

```
class add
{
Static void add(){
    int a=4;
    int b=2;
    int c=6;
    int d= a+b+c;
    sop(d);
return;
} psum(str[] args)
  {
sop (sum is);
    add();
  }
}.
```

o/p sum is
12.

- Final - when a final keyword is used in front of a variable than the value is fixed & cannot be changed.

HW Write a program to calculate the areas using methods.

Triangle	area = $\frac{1}{2} \times b \times h$.
Square	Area = a^2
rectangle	area = $w \times h$.
Parallogram	$b \times h$
Trapezoid	$\frac{1}{2}(a+b) * h$
Circle	πr^2
Ellipse	Area = πab
Sector	Area = $\frac{1}{2} r^2 \theta$.

Scanned with OKEN Scanner

papergrid

Date / /

Q Circle

```
class circle
{
 static void area()
  {
     int R = 10;
    final double Pi = 3.14;
      double area = R * Pi;
       sop(area);
    return;
  }.
 psvm(str[] args)
 {
    sop("area");
      area();
  }
}
```

o/p - area
31.4.

Q WAP to add 2 no with method with parameters.

```
class sample
{
 static void add(int a, int b)
  {
      int c = a+b;
       sop(c);
  } static void sub(int a, int b){
        int d = a-b;
         sop(d);
    }
  psvm(str[] args){
      add(10,20);
      sub(20,10);
    }
}
```

Scanned with OKEN Scanner

```
- class sample
  {
                                  localblock
  Static void order (int qty) //parameter//
  {  int price =10;                  formal arguments.
     int res= qty * price;
        Sop(res);
  }
  PSVm(str [] args)
  {
        order(5); // arguments
        order(6); // actual arguments
  }
}
```

Syntax for arguments-

(datatype, variable_name)

eg - (int a)

(int x, double y, char z)

① formal arguments-

these are the arguments which recieves the input.

② actual argument-

these are the arguments are the values given to the formal arguments.

Q WAP to give multiplication of 3no's

```
- class multiply
  {
  Static void mul (int a, int b, int c)
    {
        int d= a * b * c;
        Sop(d);
    }
```

Scanned with OKEN Scanner

```
psum(Str[] args){
    mul(10,20,30);          O/P = 600
  }
}
```

Q WAP Area of circle with method with parameter

```
class Area
{
Static void area(int r){
{
final double pi = 3.14d;
    double area = Pi * R * R;
        sop(area);
  } psum(Str[] args)
    {
        area(10);
    }                       O/P = 31.4.
  }.
```

HW Q Write 8 programs to calculate the areas with method with parameters.

Q Write a program to multiply 3 no using methods.

```
- classmultiply
  {
  Static int mul(){
      int a = 10;
      int b = 20;
      int c = 30;
    int d = a * b * c;
  return d;
} psum (str [] args){
      int y = mul();          O/P = 600
        sop(y);
  }
}
```

Scanned with OKEN Scanner

papergrid

Date: / /

Q. WAP for addition of 3 no's using method with return type.

```
Class Addition
 {
 Static int add()
 {
    int a=2;
    int b=3;
    int c=4;
    int m= a+b+c;
    return m;
  }
  PSVM(Str[] args)
  {
     int z= add();
     Sop(z);
  }
}
```

Q. Write a program to find area of circle with method with return type.

```
Class circle
 {
 Static double area() {
   int r=10;
  final double pi = 3.14d;
        double x = r*r*3.14d;
        return x;
   }
  PSVM(Str[] args)
  {     double y=area();
          Sop(y);
  }
```

Scanned with OKEN Scanner

Write 8 programs to calculate area with method with return type.

In a class we have 3 members
① variable - It is used to store data
② methods - It is used to perform some operations
③ constructor - It is used to initialize variables

- Variables and methods can be classified into static & non static
- constructor is always non static

```
class Demo
{
  variable } static
  method   } non static
  constructor - always non static
}.
```

Static -

- any member declared with the keyword static is called as static member of the class.
- static is always associated with class.
- static is on copy.
- all static members will get stored in static pool area.
- whenever we want to access static members from one class to another class will have to use,

class name.method name
or
class name. variable name.

Scanned with OKEN Scanner

Note - Whichever class is having the main method that class name should be the file name.

for each an every class there will be a class file generated.

```
class sample
{
}
class demo
{
    psvm()
}
class test
{

}
```

demo.Java.

Q WAP to add two no's with a method as static between the class

```
class sample
{
  static void Add(){
      int a=10;
      int b=20;
      int c=a+b;
        sop(c);
    }
  }
Class Demo
{
  psvm(Str[] args)
  {  sample.add();
  }
}
```

Scanned with OKEN Scanner

papergrid

Date: / /

Q Class Sample1

```
{
Static void circle()
{
 final double Pi = 3.14d;
        int R = 2;
    int K = R* R* Pi;
        Sop(K);
  }
}
Class demo1
 {
  PSVM(Str[] args)
 {
    Sample1 . circle();
  }
}
```

HW

Q Write 8+8+8 to calculate areas between the classes.

- class sample1 with R Type

```
{
 Static double area(){
  final double pi = 3.14d;
        int R = 20;
        double K = Pi * R * R;
      Return K;
    }
}  Class demo1
    {
    PSVM (Str [] args)
    double y = sample1.area();
        Sop (y);
  }
}
```

Scanned with OKEN Scanner

Non static

- Any member declared without the keyword static is non static members of the class
- Non static is always associated with object
- Non static is multiple copies
- all the Non static members will get stored in Heap memory.
- Whenever we want to access from non static to static we have to use - ~~object~~ -

```
object.variablename;
        or
object.methodname();
        or
ref variable.methodname();
```

Syntax for object creation -

```
new classname();
```

operator — object — constructor

N N S directly we can access
S N N

we need to create object.

exp - class Sample

```
{
  int a;
  void disp(){
  {
    Sop("Hello");
}
```

Scanned with OKEN Scanner

```
psvm(String[] args)
{
  Sop(new sample().a)
      new sample().disp();
  }
}
```

Q WAP to add two no from nonstatic to static.

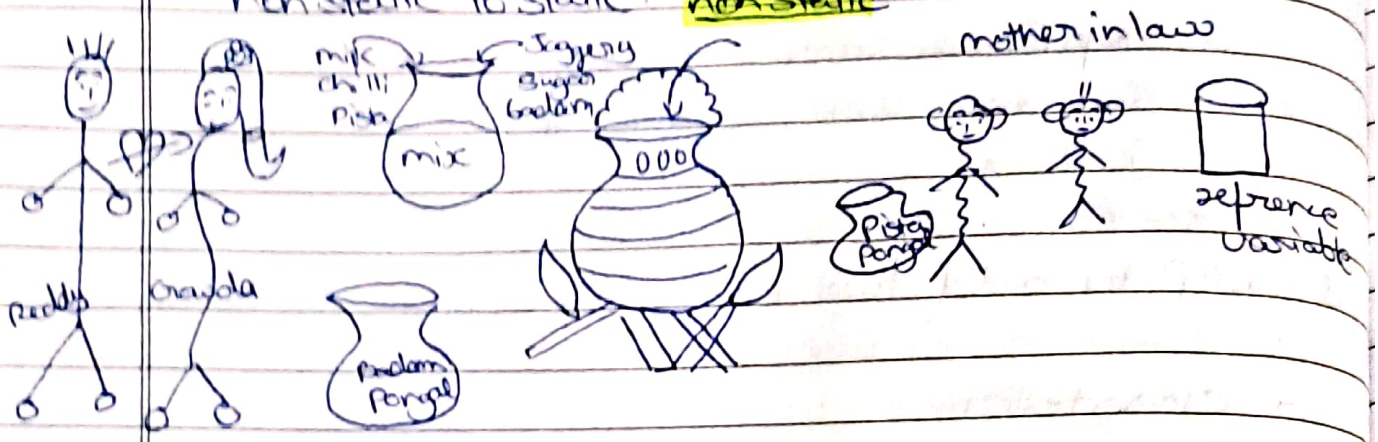
```
Class tester
{
  void add()
  {
    int a=10;
    int b=20;
    int c=a+b;
    Sop(c);
  } psvm(String[] args)
      {
        new tester().add();
      }
  }.
```

Q WAP to calculate area of a circle from nonstatic to static.

```
Class circle{
  void area()
  {
   final double pi=3.14;
      int R=2;
      double P=R*R*Pi;
      Sop(R);
  } psvm(String[] args){
      new circle().area();
  }
}
```

Scanned with OKEN Scanner

Hw Write 8 programs to calculate the areas from nonstatic to static. nonstatic



Q WAP to Access from nonstatic to static to calculate area of circle with parameter

```
- class Sample
  {
  void area(int r)
    {
    final double pi = 3.14d;
        double x = r * r * pi;
            sop(x);
    }
  PSVM (str [] args)
  {
   new sample().area(5);
  }
  }
```

Hw Write 8 programs to calculate the area with method with parameter from non static to static.

Scanned with OKEN Scanner

papergrid

Date: / /

Q

```
Class circle
{
 double area()
   {
   final double pi = 3.14d;
          int r = 10
          double K = Pi * R * R;
          return K;
   }
   Psum(str[] args)
    {
    double x = new circle().area();
       Sop(x);
    }
   }.
```

JVM memory

It is present inside RAM

```
Class Sample
 {
  void area()
  {
   final double pi = 3.142;
          int r = 5;
   double res = Pi * r * r;
          Sop(res);
 } Psum(Str [] args){
      new sample().area();
       new sample().area();
    }
  }
```

How non static is executed

Scanned with OKEN Scanner

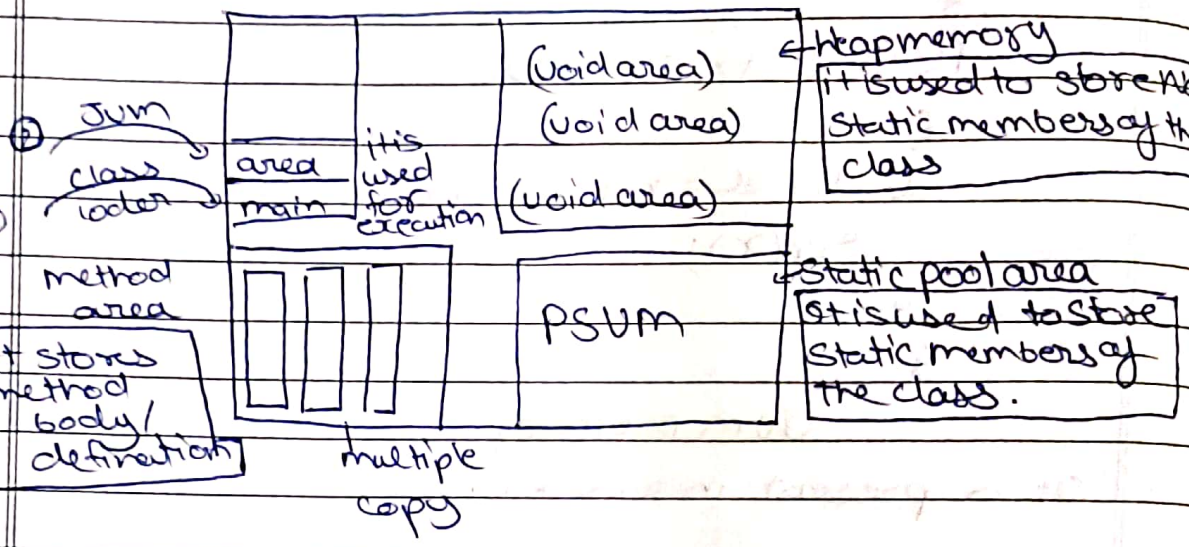
new classname()

operator — object — Constructor

It will create random memory space into the heap memory

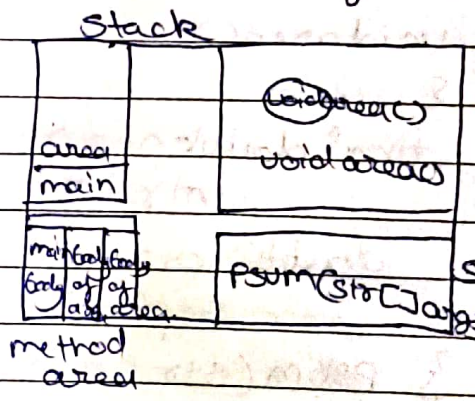
It will initialize all the non static memb into the heap memory

JVM memory



Q WAP to calculate multiplication of 3 Nos from using non static to static metod. JVM diagram.

```
Class sample
{ void area(){
    int a=2;
    int b=3
    int c=4
    int d= a*b*c;
    Sop(d);
  }
PSVM(str[]args)
  {
  new sample()area();
  new sample()area();
}
}
```



Referance Variable

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

Syntax-

① for Referencevariable declaration-

```
classname referance-variable;
        sample S1;
```

② Referance variable initialization-

Syntax - ref variable = object;

```
S1 = new sample();
```

③ On one line-

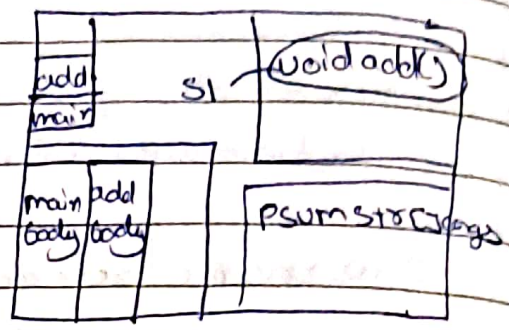
```
sample S1 = new sample();
```

Q

```
Class sample
{
Void area()
{
 final double pi = 3.14d;
   int r = 5;
  double res = pi * r * r;
        sop(res);
 }
 PSVm(str[] args){
    sample S1 = new sample();
        S1. area();
    }
}
```

Scanned with OKEN Scanner

WAP to add two nos from nonstatic to Static through ref variable.




```
Class Cool
{
 Void add()
 {
 int a=2;
  int b=3;
  int c= a+b;
  Sop (c);
 }
 Psum (str[] args)
 {
  Cool S1= new cool();
     S1.add()
 }
}.
```

Q WAP for multiplication of 3 No's using ref variable nonstatic to static.

```
class demo
 { void mul()
  { int a=1;
      int b=2;
      int c=3;
      int d= A*b*c;
        Sop(d)
  } Psvm(str[] args){
      demo d1= newdemo();
            d1.mul();
    }
  }
```

Scanned with OKEN Scanner

homogeneous type of object creation when the name of ref variable & class name is same.

HW write 8+8+8 program to access from non static to static through ref variable.

- class demo
{
void area(int r)
{
final double pi = 3.14
double res = Pi * R * R;
Sop(res);
} psvm(str[] args){
demo d1 = new demo();
d1.area(2);
}
}.

- with return type.

class circle
{
double area(){
int r = 2
final double pi = 3.14d;
double res = pi * R * R;
return res;
}
Psvm(str[] args)
{
circle c1 = new circle();
double a = (c1.area());
Sop(a);
}
}

Scanned with OKEN Scanner

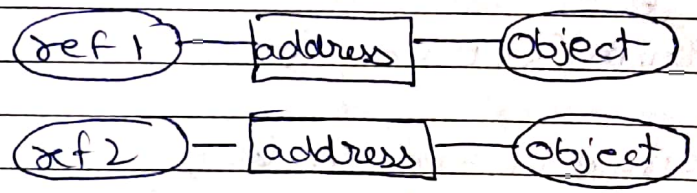
Q WAP to print reference variable

whenever we print a reference variable it will print the object address i.e fully qualified path.

Syntax -
Package name . classname @ hexadecimal-no
fully qualified path
o/p in the command prompt will be classname@ hexadecimal no so dont worry.

Q - Class sample1
{
PSVm(str[] args)
{
Sample1 s1 = new sample1();
Sample2 s2 = new sample2();
Sop(s1);
Sop(s2);
}
}

We will get 2 address one for s1 & one for s2
o/p - sample1 @ 15d69742
sample1 @ 6d06d69c.



```
class sample
{
 psvm(Str [] args)
  {
   Sample S1 = new sample1();
        sample S2 = S1;
         Sop (S1);
         Sop(S2);
    }
  }
```

o/p - Sample @ 15 d69742
Sample @ 15 d69742

When to go for static & when to go for nonstatic

- use static for things that are the same for everyone (shared information)
- eg - imagine you want to keep a track of total no of cars produced by a company. the total count is the same for all the cars and dosnt depend on any specific car type.

- use nonstatic for thing that is specfic to an individual (personal data)
eg - consider individual cars with their own unique characterstics like color, speed, mileage each car has specfic data that are not shared among all the car.

if we make changes in the total count the for All the cars count will change, whereas in case of nonstatic if we change color of one car then only one cars color will get changed. as we will creat object for that car in case of non static.

Scanned with OKEN Scanner

Static & non static -

Real time examples on static & non static -

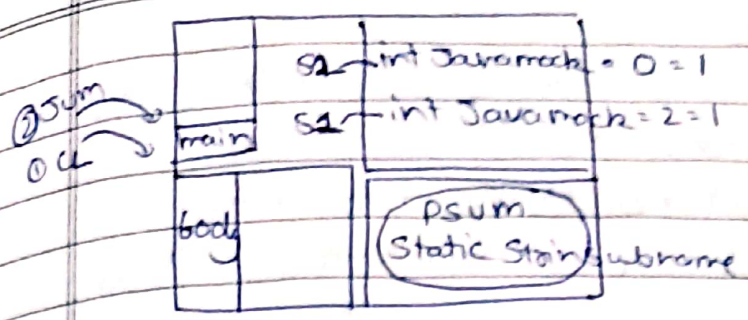
- Sanju & Geeta Joined Jspiders in the first mock of JAVA geeta got the mock rating 1 Sanj got the mock rating 2 sanju was frustrated late evening Sanju met his friend he gave an idea study hard whole night & take remock Sanju took the remock & he got the mock rating as 1

Q WAP to display Sanju & Geeta's mock rating

HW Write 10 programs to decide when to go for static & non static.

```
Class Student
{
int Javamock;
Static String sub name = "Java";

PSVM (Str[] args)
{
Sop("subject name is" + sub_name);
    Student s1 = new student()
        s1. Jayamock = 2
        Sop (s1. Javamock)
    student s2 = new student();
        s2. Javamock = 1
    sop (s2. Javamock);
        s1. Javamock = 1
        Sop (s1. Javamock);
    }
}
```



Q WAP to calculate area of circle from non static to static between the classes.

Ans WAP 8+8+8 programs to access from non static to static between the classes.

```
class sample1
{
 void area(){
  final double pi = 3.142;
        int r=5;
   double res = Pi * R * R;
        Sop(res)
   }
 } class Demo1
  {
  Psvm(str[] args){
  {
  Sample S1 = new sample()
    S1.area();
   }
  }.
```

CLASS

class is a blueprint or tempelate to create object.

Scanned with OKEN Scanner

Object

object is a realtime entity which has its own state & behaviour

- State is nothing but nonstatic variables.
- behaviour is nothing but nonstatic method.

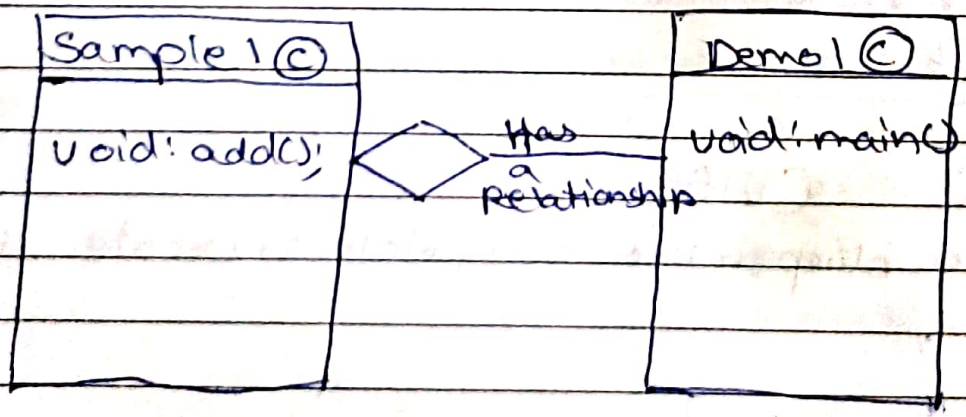
- State defines what data it can store
- behaviour defines the way it can behave.
- whenever we create an object it will have both state and behaviour
- the object address will get stored in a reference variable.
- multiple object address can be stored in multiple reference variable
- an object address can be stored in multiple reference variable.

Composition -

- a class having an object of another class is called as composition.
- it is also called as has a relationship

Class diagram

It is a pictorial representation to represent the members of the class.



Scanned with OKEN Scanner

```
class Sample1
{
    void add()
    {
     int a=10;
      int b=20;
      int c= a+b;
          Sop(c);
    }
} ClassDemo1
{
    PSVM (Str[] args)
    {
    Sample1 S1= new Sample1() <
          S1.add;
    }
}
```

composition.

Inheritance

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

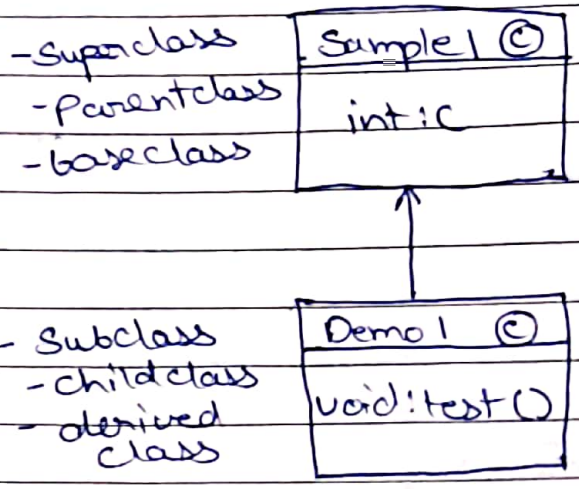
- inheritance is also called as Is a relationship.

in Inheritance we have 5 types.

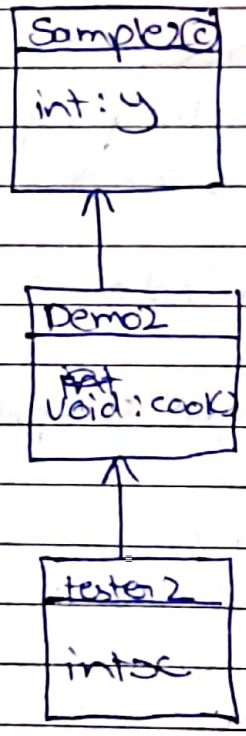
1. Single level inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. hybrid inheritance.

Scanned with OKEN Scanner

- **Single level Inheritance** - a subclass inheriting the properties from only one superclass is called as single level inheritance.

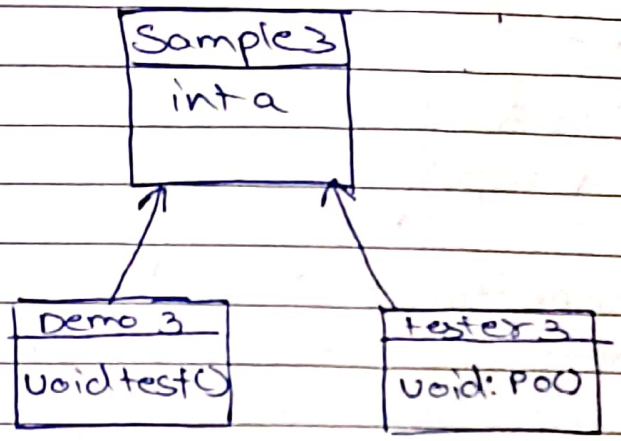


- **Multilevel Inheritance** - a subclass inheriting the properties from its superclass inturn superclass inheriting the properties from its superclass is called multilevel inheritance.



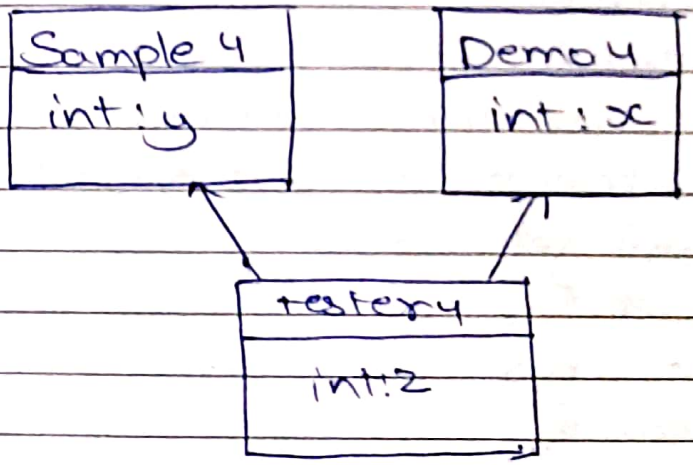
Scanned with OKEN Scanner

- **Hierarchical inheritance** - multiple subclass inheriting the properties from only one superclass or common superclass is called as hierarchical inheritance.



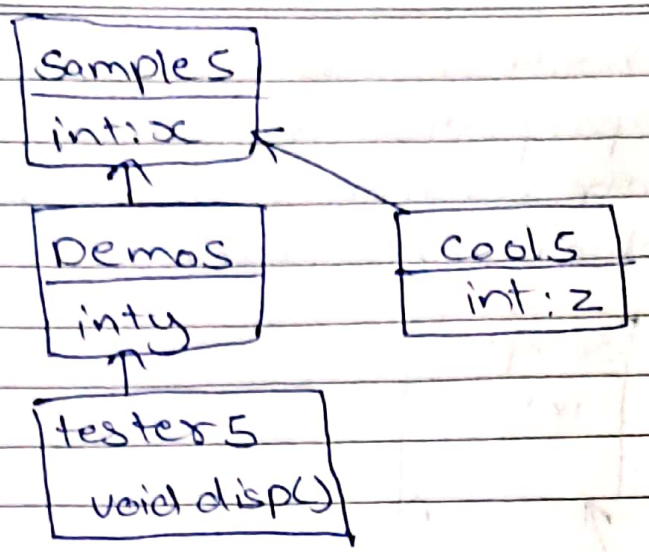
- **Multiple inheritance** - a subclass inheriting the properties from multiple superclasses is called as multiple inheritance.

- we cannot achieve multiple inheritance through classes but can achieve through interface.



- **hybrid inheritance** - It is a combination of single level, multilevel & ~~multiple~~ hierarchical inheritance is called as hybrid inheritance.

Scanned with OKEN Scanner



We use inheritance so that if we have written a program we can reuse it in the current program i.e code reusability.

Constructor

Constructor is a special method or special member of the class which is used to initialize variables.

Rules - ① Constructor name should be same as classname
② Constructor will not have any return type
③ Constructor will not return any value
④ Constructor is always non static
⑤ Wherever an object is created constructor will get invoked.

Syntax -
```
Class classname
{
    classname(){
    {
        ==        statement
    }
    return;//optional
}
```
constructor.

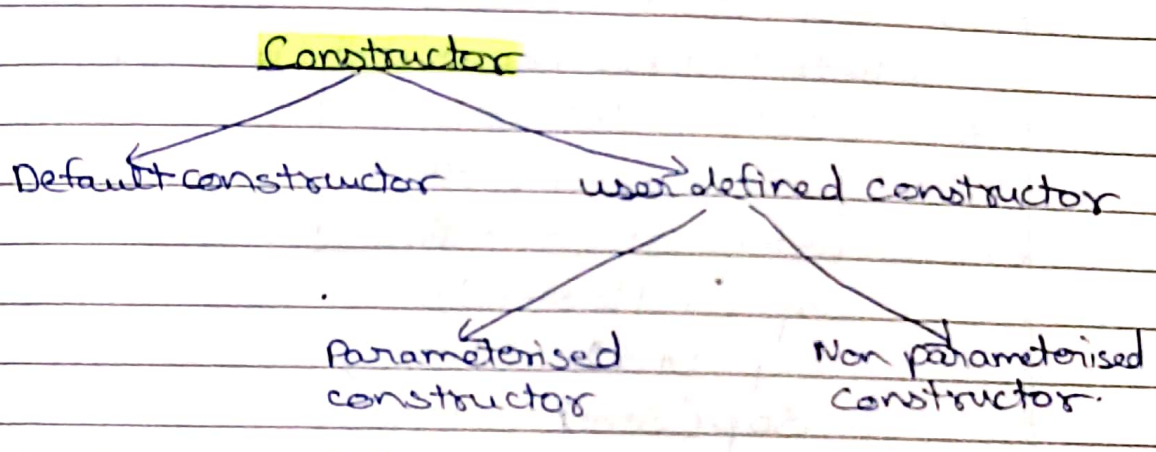
Scanned with OKEN Scanner

papergrid

Date: / /

```
Class Sample
{
 sample()
 {
  Sop("hey i am constructor");
  return;
 }
 PSVM(str [] args){
  sop(" main starts---");
   new sample();
   sop("main ends");
 }
}
```

o/p- mainstarts--
hey i am constructor
main ends.



Constructor are of 3 types-
Default constructors
Parameterised constructors.
Non parameterised constructors.

Real time example on constructors.
WAP to initialize student Id student name & section using constructors.

Scanned with OKEN Scanner

papergri
Date: / /

Q

```
Class Student
{
 int stdid;
 String Stdname;
 char stdsec;
 Psvm (Str [] args)
 {
   Student Std1 = new student();
        Std1.stdid = 123;
        Std1.stdname = "Raju";
        Std1.stdsec = 'A';
   sop(std1.stdid);
   sop(std1.stdname);
   sop(std1.stdsec);
   Student Std2 = new student();
        Std2.stdid = 124;
        Std2.stdname = "RANI";
        Std2.stdsec = 'B';
   sop (Std2.stdid);
   sop(Std2.stdname);
   sop(Std2.stdsec);
 }
}
```

if we have 100 variab
we will have to write 10
times or if we have t
reinitialize then we ha
to write it again whi
will increase the code
size. So this was the
drawback to avoid
it constructors are
used.

```
Class student
{
  int stdid;
  String stdname;
  char stdsec;
 Student (int a, string b, char c)
  {     stdid = a;
        stdname = b;
        stdsec = c;
  return;
 }
```

Scanned with OKEN Scanner

```
PSVM(str[] args){
 {
 student std1 = new student(123,"Raju",'A');
        sop(std1.stdid);
        sop(std1.stdname);
        sop(std1.stdsec);
 student std2 = new student(124,"RAMI",'B')
        sop(std2.stdid);
        sop(std2.stdname);
        sop(std2.stdsec);
      }
 }
```

O/P - (std1) int stdid = ~~0~~ → 124
~~int~~ String name = ~~null~~ → RAMI
char stdsec = ~~'\u000'~~ → A

(std2) int stdid = ~~0~~ → 124
String stdname = ~~null~~ → RAMI
char stdsec = '\u0000' → 'B'

Q WAP to initialize employee id, employee name, employee salary.

```
- Class employee
  {
  str employee_name;
  int employee_Id;
  int employee_salary;
     employee (int a, int b, str c){
        employee name = c;
        employee ID = b;
        employee Salary = a;
        return;
      }
  PSVM(str[] args){
  employee E1 = new employee(1000, 1234, Raj);
```

```
Sop (E1.employee name);
Sop(E1. employeeId);
Sop(E1. employee salary);
employee E2 = new employee(1001, 1235, Rohan);
  Sop (E2. employeeame);
  Sop(E2.employeeId);
  Sop(E2. employeesalary);
 }
}
```

Q Mobile model name, cost, & mobile Grade -

```
- Class Mobile
 {
  Str mobilename;
    int cost;
  char Grade;
  Mobile (int a, str b, char c);
   {
        mobile name = b;
        cost = a;
        Grade = c;
  } PSVM (Str[] args) {
  Mobile m1 = new mobile (1000, Realme, c);
    Sop (m1. mobilename);
    Sop (m1. mobilecost);
    Sop (m1. Grade);
  Mobile m2 = new mobile (2000, Realme, c);
      Sop(m2. mobilename);
      Sop(m2. mobilecost);
      Sop(m2. Grade);
  }
}
```

Scanned with OKEN Scanner

Write 10 programs to initalize the variables through constructor.

```
Class lulumall
{
 string mall_name;
 int mall_capacity;
  char mallgrade;
 lulumall (String a , int b , char c) {
            mallname = a;
             mall capacity = b;
             mall grade = c;
   } psvm (string [] args) {
   lulumall l1 = new lulumall ("lulumall", 5000, 'A');
         sop(l1. mallname);
          sop(l1. mall capacity);
          sop(l1. mall grade);
    }
}
```

Write 10 programs to initalize the variables using this keyword.

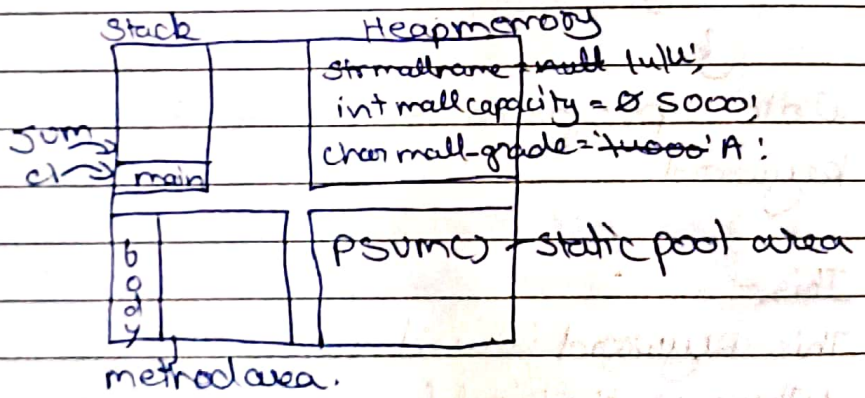
This:-

This keyword is used to point to current object, whenever the local & global variable names are same to differentiate between them we use the this keyword.

this keyword can be used only in the non static context
this keyword is also called as default reference variable.

Scanned with OKEN Scanner

```
Class lulumall
{
String mall-name;
int mall-capacity;
char mall-grade;
lulumall(string mallname, int mall-capacity, char mall grade)
    {
        this.mallname = mallname;
        this.mallcapacity = mall-capacity;
        this.mallgrade = mall-grade;
    }
PSVM (str[] args){
    lulumall l1 = new lulumall("lulu", 5000, 'A');
        Sop(l1.mallname);
        Sop(l1.mallcapacity);
        Sop(l1.mallgrade);
    }
}
```



Pass by value

Calling or invoking the method by passing primitive type of data is called as pass by value.

```
Class Demo
{
static void add (int a)
 {
   Sop (a+a)
 }
P sum (str [] args)
 {
    int x = 90;
      add(x);
  }
}
```

int x = 90
int a = x

Pass by reference - calling or invoking the method by passing reference variable is called as pass by referance.

pass by referance is also called as pass by value.

```
Class sample
{
 int x = 90;
 static void disp (sample s2)
  {
      Sop (s2.x);
  }
 PSVM(String [] args)
  {
   Sample S1 = new sample();
        disp(S1);
  }
}
```

S2 = S1

Scanned with OKEN Scanner

```
class demo
{
static void add()
{ int a=10
}
PSVM(str[]args)
{ add();
}
}
```

```
Class Demo1
{
Static void add (int a)
{
}
PSVM (str[]args)
{ int a=3;
add(a)
}
}
```

Pass by value.

```
class demo2
{
static int add ()
{
return 10;
}
PSVM (str[]args)
{
int x = add()
sop(x)
}
```

Local

```
Class Demo 3
{
PSVM ( )
{
int a=30;
}
}
```

Global.

```
class sample
{
static int x = 90
PSVM( ){
}
}
```

```
class cool
{
static void add ()
{
cool a=new cool();
}
PSVM(str[]args)
{ add();
}
}
```

Reference variable using method.

```
Class cool1
{
Static void add(cool c2)
{
}
PSVM( str[] args)
{
cool c1= new cool();
add(c1);
}
}
```

Pass by reference.

method with parameter

```
Class cool2
{
Static cool2 add()
{
return new cool2();
}
PSVM(str[]args)
{
cool2 x = add();
}
}
```

return type.

```
Class cool3
{
PSVM()
{
cool3= new cool()
}
}
```

```
Class cool4
{
Static cool4 x= new cool4()
PSVM(str[]arg
}
}
```

Can be classified into static & non static & Local & global.

Scanned with OKEN Scanner

papergrid
Date: / /

```
Class SBI_Bank{
    void account()
  {
  Sop("saving account money---")
  }
  class phonepe
  {
  Static void transaction(sbi_bank s2)
        {
        s2.account()
        }
  }
  class customer.
  {
   SBI_bank s1 = new sbi_bank();
          s1.account();
      phonepe.transaction(s1);
    }
  }
```

HW Write 5 programs with realtime examples on pass by reference.

Method Overloading.

Developing multiple methods with the same name but variation in the arguments lists is called method overloading.

Variation in the arguments list means-

1. Variation in the length of the arguments.
2. Variation in the order of occurence of the arguments
3. Variation in the data type.

Scanned with OKEN Scanner

Rules-

1) The method name should be same.
2) there should be variation in argument list.
3) there is no restriction on access specifier modifier & returntype.
4) We can overload both static & non static methods.

- When do we go for method overloading?
Whenever we need to perform common task or operation, then we should go for method overloa

- time complexity
- bluissness.

```
Class whatsapp
{
 void send(int no)
  {
   Sop("Sending no" + No);
  }
 void send(string texting)
  {
   Sop("sending texting" + texting);
  }
 void send(int no, string texting)
  {
   Sop("sending no" + no + "  " + texting);
  }
 void send(string texting, int no)
  {
   sop("sending text & no" + texting + no);
  }
}.
```

Scanned with OKEN Scanner

```
Class mainclass
{
psvm(str[] args)
{
whatsapp w1 = new whatsapp();
    w1.send(123, "Hi")
    w1.send(26);
    w1.send("Hi");
    w1.send("Hello", 96);
  }
}
```

HW/ Q Write 3+3 programs with real time example on method overloading.

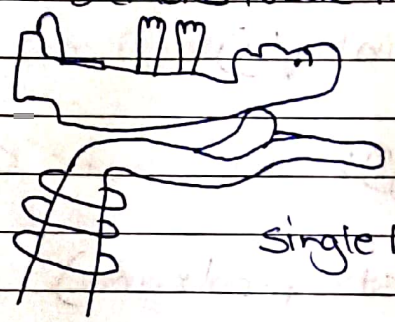
```
Class amazon
{
Static void buy (int qty)
  {
  Sop("buy by qty");
  }
Static void buy (str prdname)
  {
  sop("buy by productname");
  }
static void buy (int qty, str prdname)
  {
  Sop("buy by qty & prdname" + qty + prdname);
  }
Static void buy (string productname, int qty)
  {
  Sop("buy by prdname, qty");
  }
}
```

Scanned with OKEN Scanner

```
Class mainclass
{
PSVM(str[] args)
{
Amazon.buy(5);
Amazon.buy("nike");
Amazon.buy(4,"Puma");
Amazon.buy("nike", 5);
}
}
```

Note - If the method parameter is of same length and same data type then we get compile time error.

- WAP to demonstrate inheritance -



Single Level Inheritance.

multiple inheritance.

Multilevel inheritance.

K_1

K_2

K_3

K_4.

Scanned with OKEN Scanner

papergrid

Date: / /

Q WAP to demonstrate inheritance.

```
class Sample
{
 int c = 80
}
class Demo extends sample
 {
  void test()
  {
   sop("hey its test");
  }
 }
class mainclass
 {
 psvm (str[] args)
 { demo d1 = new Demo()
     s.o.p (d1.c);
       d1.test();
  }
 }
```

- Class Sample2 { multilevel inheritance.
 int y=10;
 }
 Class Demo2 extends sample2
  {
   void cool()
   {
    Sop ("hey its test");
   }
 } Class tester2 extends Demo2
  {
    int x = 10;
  }

```
Class mainclass
{
PSum (Str [] args)
{
tester2 t1 = new tester2();
        Sop (t1.y);
        t1.cool();
        Sop(t1.x);
  }
}.
```

Q When do we go for inheritance-

- Code reusability

```
Class sampleC
 {
 int a = 10;
 }
 Class Demo3 extends sampleC.
  {
  voidtest()
  {
   Sop("hello");
  }
}Class ~~mainclass~~ tester3 extends SampleC
  {
   void fo()
   {
    Sop("Hi");
   }
}Class mainclass
  {
```

```
psvm (Str[] args)
{
  Demo D3 = new Demo()
    sop(D3.a);
      D3.test();
  tester3 t3 = new tester3()
      sop(t3.a);
        t3.fo();
  }
}
```

Array-

Array is a linear data structure which is used to store homogeneous type or same type of data.

Drawbacks of array-

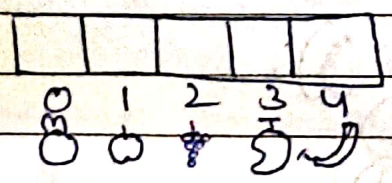
1) Size is fixed
2) we can store only homogeneous type of data.

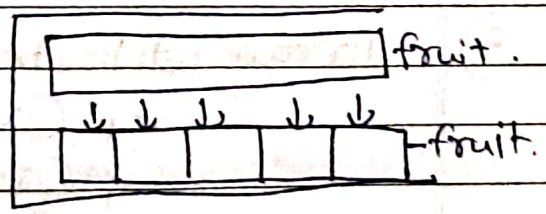
We can create array of two type-

1) with new operator
2) without new operator.

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

Fruit-





1) Array declaration

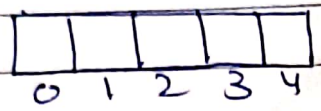
Syntax - datatype [ ] Array Name;

eg:- int [ ] arr;

2) Array size initialization

Syntax = Arrayname = new datatype [size];

arr = new int [5];



3) Array storing the data into an array -

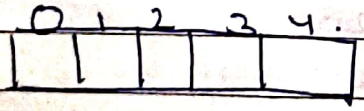
Syntax- Arrayname [index] = value;

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

4) Array Declaration & size Initialization in a Sing line.

Syntax - datatype [ ] arrayname = new datatype [s

int [ ] arr = new int [5];



5) Array utilization -

sop (arr [0]);
sop (arr [1]);
sop (arr [2]);
sop (arr [3]);
sop (arr [4]);

Scanned with OKEN Scanner

Sop (arr[5]);
└ array Index out of bounds exception.

If we don't store any value in a arrays if we try to print we will get default value of array type

⑥ to find size of an array
Syntax: arrayname.length.
Sop (arr.length);

⑦ code optimization -
HW # write 9+9 programs on array with new operator & without new operator.

```
for(int i=0; i<arr.length; i++)
  {
    Sop (arr [i]);
  }
```

⑧ Array value Reinitalization -
syntax - arrayname [index] = newvalue;

arr [3] = 90;

⑨ Array Declaration & Storing data directly

Syntax - Datatype[] arrayname = {v1, v2, v3 - -};

ex int[] arr = {10, 20, 30, 40}

When we know the data & size then we go for this.

Scanned with OKEN Scanner

\t = tabspace

paperg

Date: / /

(10) Coping the value from one array to another arr

```
int[]arr={10,20,30,40,50}

    int[]abb= arr;
```

Q

```
Class Sample3
{
PSVM(str[]args)
{
Boolean[]arr={true,false,true,true};
  Sop("***  **")
  Sop("index \t values");
  Sop("+***  ***");
 for(int i=0; i<arr.length; i++)
  {
   Sop(i+"\t"+arr[i]);
  }
 }
}
```

O/p

```
**  **  **
index        value
**  **  **
0            true
1            false
2            true
3            true.
```

Q

```
Class Sample3
{
PSVM(str[]args)
{
int[] arr = new int[5]
    arr[0]=10;
    arr[1]=20;
```

Scanned with OKEN Scanner

```
arr[2] = 30;
arr[3] = 30;
arr[4] = 40;
sop ("* * * * *");
sop ("index \t values");
sop ("* * * * *");
for (int i=0; i< arr.length; i++)
    {
        sop (i + "\t" + arr[i]);
    }
}
}
```

Method overriding -

Developing a method in the subclass with the same name and signature as in the superclass but with the different implementation in the subclass is called as method overiding.

Rules-

① The method should be non static
Note - (only Non static can be inherited static cannot be inherited)
② there should be is a relationship.
③ the method name & signature should be same in the subclass as in the superclass.
- we can override only non static method but we cannot override static method.

#

Note for understanding - we can inherit both static & non static but to avoid confusion we say we inherit non static. we can call a static method/variable with the object but it will make things get confused. as static is always associated with the class. we call static method using class name.

Scanned with OKEN Scanner

Q What will happen if I try to override static.

- We cannot override static method because it is always associated with the class, not with individual objects or instances. ~~i.e its scope is in~~

realtime example - think in your family there is a rule how everyone should say "hello" (a static method in the class) now one day your little brother decides to make his own rule for saying 'hello' (a new static method in subclass) But now ~~how~~ no matter how much your little brother insists on his new rule the original family rule doesn't change. if someone asks you to someone asks you to say "hello" following the family rule you can't suddenly use ~~the~~ your little brother's rule. His rule doesn't replace the family rule its just a new rule for a specific situation (subclass).

If you declare a static method in a subclass with the same name and signature as in super class it won't actually override the superclass method. Instead it will hide it. this means that the subclass method is unrelated to the superclass method & wont participate in overriding.

example -
```
class whatsapp v1
{
    void calls()
    {
        sop ("call to1 person");
    }
}
```

```
Class whatsapp-v2 extends whatsapp-v1
{
 void calls()
 {
 sop ("call to 5 people);
 }
class mainclass.
{
psvm( str[] args)
{
 whatsapp v2 v2 = new whatsappv2();
        v2. calls();
  }
}
```

we mainly use method overiding for updation.
to prevent downtime.

wsp with Real time example on method overriding.

```
Class OLA-V1
{
 void ride()
 {
  sop ("ride with cabs");
 }
} class OLAV2 extends Ola V2
 {
  void ride()
  {
  Sop ("Ride with auto and cabs");
  }
}
Class mainclass {
```

Scanned with OKEN Scanner

```
Public static void main (Str[] args)
{ OlaV2 V2 = new OlaV2 ();
    V2.ride();
}.
```

Super keyword = Super keyword is used in the case of method overriding along with the subclass implementation if we need super class implementation the we will use

Super . methodname.

Super key word is non static we cannot use it with static method.

Method Hiding.

In case of method overriding the superclass implementation is hidden by the subclass implementation & this concept is called as method hiding.

```
Class Instagram V1
{
Void post()
  {
   S.O.P ("post any images");
  }
} class Instagram-V2 extends Instagram V1
  {
    void post() {
     S.O.P ("post videos");
     Super.post();
    }.
  }
```

Scanned with OKEN Scanner

```
Class mainclass
{
 Psvm (Str[] args){
  Instagram U2 = U2 .Instagram U2();
          U2.post();
  }
}
```

Type Casting:-

converting from one type to another type is called as type casting.

on type casting we have
1. Primitive type casting
2. class typecasting / derived type casting.

Primitive type casting:-

* widdening — implicitly / explicitly
* Narrowing — explicitly.

Class type casting:-

* upcasting — implicitly / explicitly
* Downcasting — explicitly.

- Primitive typecasting:-
converting from one primitive datatype to another primitive datatype is called as primitive typcasting.

widdening- converting from smaller primitive datatype to any of its bigger primitive datatype is called as widdening.

Scanned with OKEN Scanner

Widdening can be done both implicitly & explicitly.

② Narrowing - converting from bigger primitive datatype to any of its smaller primitive data is called as Narrowing.

Narrowing can be done only explicitly.

implicity	explicitly
① double x = 20;	① double x = (double) 20;
sop(x); o/p = 20.0d	sop(x); o/p = 20.0d
② float y = 36;	② float y = (float) 36;
sop(x); o/p = 36.0f	sop(y); 36.0f
③ double z = 31.88f;	③ double z = (double) 31.8f
sop(z); o/p 31.88d	sop(z); o/p 31.8d.
Internally it will take the datatype & cast it.	we will have to declare what type we want the data to be casted.

widning

byte short int long float double

narrowing.

Narrowing

① int x = (int) 59.0 — we will do it explicitly because there is data loss
sop (x)

② byte b = (byte) 96.3f;
sop (x) o/p 96

③ long z = (long) 56.37d;
sop(z); o/p 56;

Eclipse

eclipse is an ide (Integrated development enviornment) in eclipse we have various versions.

advantage of eclipse:-

1) It will segregate Java & class files i.e all the Java in the source folder & class in bin folder.
2) on a single platform we can develop multiple application like XML, WEB, Java & plugins by changing the perspective.

How to download eclipse:-

- on the system.
- go to google & type eclypse neon download.
- on search results click on the first link
- click on 64 bit under Java EE Developers
- the software will be downloaded in downloads folder.

How to install ecllipse

1) go to the downloaded folder Right click & click extract all
2) double click on extracted folder & double click on eclipse folder.
1) select workspace (where the Java programs should be saved)
2) close the welcome screen.
3) select perspective as Java
4) Create project
5) create package / it is nothing but folder.
6) after package create class in the folder.

Scanned with OKEN Scanner

1) create project-
file -> new -> Java project -> project name -> expand project & selected SRC.

2) create package.
file -> new -> package -> package name -> finish

3) create class ->
select package -> file -> new -> class -> class name finis.

4) WAP & click on play button.

web -> .war
Java -> .Jar
SQL -> .Tar
} Packaging extension

Class typecasting-

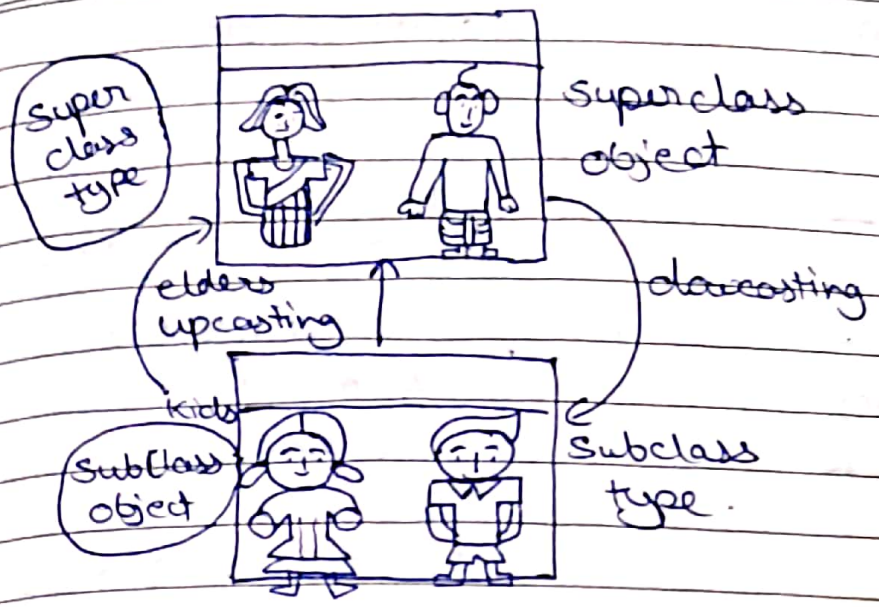
Converting from one class object to another class type is called as class typecasting.

- upcasting - converting from subclass object to superclass type is called as upcasting.

upcasting can be done both implicitly & explicity.

- downcasting - converting from superclass object to subclass type is called as downcasting.
- down casting should be done always explicitly

Scanned with OKEN Scanner



//homogeneous type of object creation.

```
Sample S1 = new Sample()
    sop(S1.x)
Demo D1 = new Demo()
    Sop(D1.x)
    d1.add().
```

upcasting-

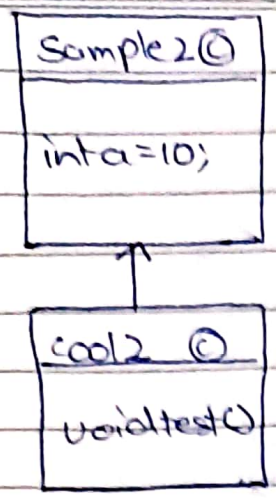
```
Sample S1 = new Demo() // heterogeneous type of
    S1.x                   object creation.
```

not possible ~~S1.add();~~
as it hides subclass properties.

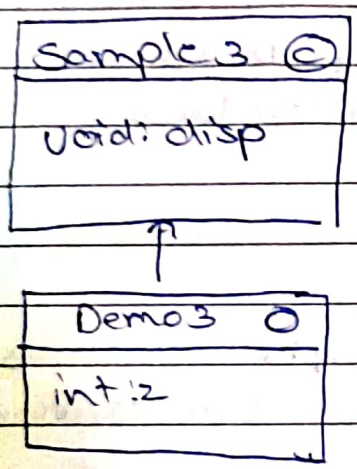
Downcasting-

```
Demo D1 = (Demo) S1;
    Sop(d1.x);
    d1.add();
```

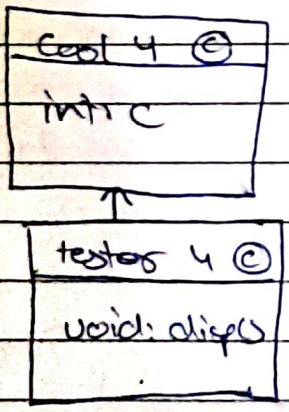
Scanned with OKEN Scanner

② 


```
class Sample2
{
int a = 10;
}
class cool2 extends Sample2
{
void test(){
a+30;
}
class main
{
PSVM(Str[] args)
upcasting Sample2 S1 = new cool2
Sop (S1.a);
downcasting cool2 c2 = (cool2) S1
Sop (C2.a)
C2.test();
```

③ 


```
PSVM (Str[] args){
Sample3 S3 = new Demo3();
S3.disp();
Demo3 B3 = (Demo3) S3;
B3.disp();
Sop(B3.2);
}
```

④ 


```
psvm(Str[] args){
cool4 c4 = new tester4()
Sop (c4.c);
tester4 t4 = (tester4) c4;
Sop(t4.c);
t4.disp();
}
```

Q Write 14 programs to perform upcasting & downcasting.

Scanned with OKEN Scanner

papergrid

Date: / /

```
Class sample3 {
  void disp()
  {
  sop ("its disp---");
  }
} class Demo3 extends sample3
  {
    int z=90;
  }
  Public class mainclass {
  PSVM (Str[] args){
    Sop ("upcasting);
    Sample3 S3 = new Demo3();
        S3.disp();
    Sop ("downcasting * * *");
    Demo3 D3 = ~~(Sample3)~~ (Demo3) S3;
        Sop (D3.z);
        D3.disp();
  } }
```

Abstract class

- concretemethod-

any method which has both declaration & defination is called as concrete method.

```
void disp()
{
  ____
  ____
  ____
}
```

- concrete class

any class which has only concrete methods is called as concrete class.

```
Class Sample {
  void disp()
  {
```

Ctrl+shift+f.

papergrid

Date: / /

- Abstract method-
Any method declared with the keyword Abstract is called as abstract method.

```
abstract void disp();
```

- Abstract method do not have a body.

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

```
abstract class sample
{

}.
```

- If a class consist of abstract method then the class should be declared as abstract but viceversa is not true.

```
abstract class Demo
{
abstract void test()
}.
```

- An abstract class can have both abstract & concrete method.

```
abstract class sample
{
abstract void disp();
void cool()
{
sop("hello");
}
```

Scanned with OKEN Scanner

the class which provides the implementations for the abstract methods is called as implementation class.

we cannot declare abstract method as static final & private.

if any one of the abstract method is not overriden in the subclass then the subclass should be declared as abstract.

we cannot create object for abstract class.

abstract class will have default constructor

```
abstract class sample1
{
abstract void disp();
abstract void test();
}
class Demo1 extends sample1
{
void disp()
{
Sop("Hello");
}
void test(){
Sop("Hi")
}
class mainclass{
Psvm(str[]args)
{
Demo1 D1 = new Demo1();
    d1.disp();
    d1.test();
}
}
```

```
abstract class sample
{
abstract void disp();
abstract void cool();
}
abstract class Demo2
        extends sample2
void disp()
{ sop("Hello");
}
//abstract void cool();
} class cool2 extends Demo2
{ // void disp
  {sop("Hello");
void cool()
{
sop("I am cool")
}
psvm(str[]args)
cool2 c1 = new cool2();
c1.~~test~~ disp();
c1.cool();
}
}
```

In case of method overriding eventhough it is upcasted we get overridden implementation.

```
Class Demo {
 int a = 10;
  void add(){
   Sop("hello");
  }
} class sample extends Demo{
      void add(){
        Sop("bye");
      }
}
Public class mainclass{
  PSVM(Str[] args)
 { Demo S = new sample()
     S.add();
  }
}
```

o/p = bye

Interface

constructor
methods
variable
blocks

- Interface is one of the keyword.
- Interface is one of the Java types.
- Interface is pure abstract body.
- In Interface we can develop only two members i.e variables & methods.
- In Interface all the variables are by default public static & final.
- In interface all the methods are by default public & abstract.
- Interface dosent support constructor so we cannot create object for it.

Scanned with OKEN Scanner

Java provides special keyword called implements to inherit the properties from interface to class.

C ↑ extends C I ↑ implements C I ↑ extends I C ↑ X I

to inherit properties from interface to interface we have to use extends.

Interface doesn't extends any class, Interface itself ~~as~~ acts as a supermost type.

```
abstract interface Demo2 {
    public static final int a = 10;
    public abstract void add();
}
public class mainclass3 implements Demo2
    public void add() {
        // public static final int a = 10;
        Sop("hello");
    }
    PSVM(Str[] args) {
        mainclass3 m = new mainclass();
        m.add;
        Sop(a);
    }
}.
```

dont compile default

papergrid

Date: / /

Q2-
```
interface Demo6{
    void add();
}
interface sample2 extends Demo6{
    int a=10
    //void add();
    void sub();
}
public class mainclass implements sample2{
    PSVM (str[] args){
        mainclass = new mainclass();
        m.add()
        m.sub()
        sop(a).
    } public void add(){
        sop("hello")
    }
    public void sub(){
        sop("bye");
    }
}
```

O/P = Hello
bye
10

Multiple Inheritance

```
interface hello1{
    void add();
}
interface hello2{
    void sub();
}
Class mainclass implements hello1, hello2{
    public void add(){
        sop("I am add");
    }
```

```
public void sub(){
sop("I am sub");
}
psvm(str[] args)
{
mainclass m1 = new mainclass();
    m1.add();
    m1.sub();
}
}
```

Polymorphism

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

Poly means many + morphism means Form

In polymorphism we have 2 types-
Compile time polymorphism (Static polymorphism)
Run time polymorphism (Dynamic polymorphism)

Compile time polymorphism = the method declaration gets binded to its defination at the compile time by the compiler based on the arguments passed is called as compile time polymorphism.

Since the method declaration get binded to its defination at the compile time hence it is also called as early binding.
Once the method declaration gets binded to its defination it cannot be rebinded hence it is called as static binding.
Method overloading is an example of compile time polymorphism.

Scanned with OKEN Scanner

papergrid

Date: / /

2) Run time polymorphism -

- the method declaration gets binded to its defination at the Run time by the JVM based on the object created is called as Run time Polymorphism.

- Since the method declaration get binded to it defination at the Runtime ~~based on object created~~ Hence it is called as Late binding.

Once the method declaration gets binded to its defination it can be rebinded hence it is called as dynamic binding.

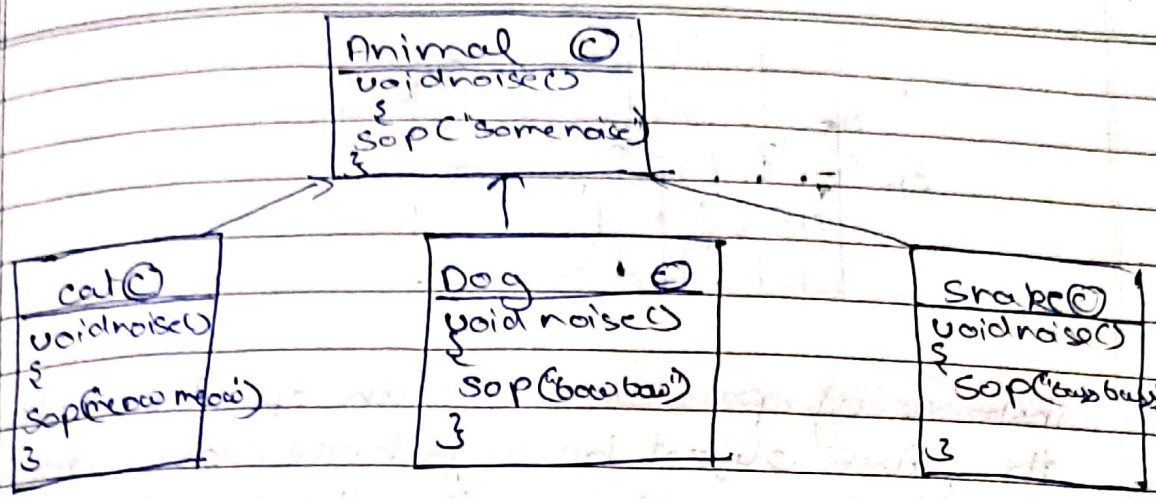
Method overiding is an example for Run time polymorphism.

Note - through Run time polymorphism we can achieve generalization, single point of contact, loose coupling - if any internal inhancement happen it will not affect the usage.

generalization treating different thing in a similar way based on what they can do even if they are of different types.

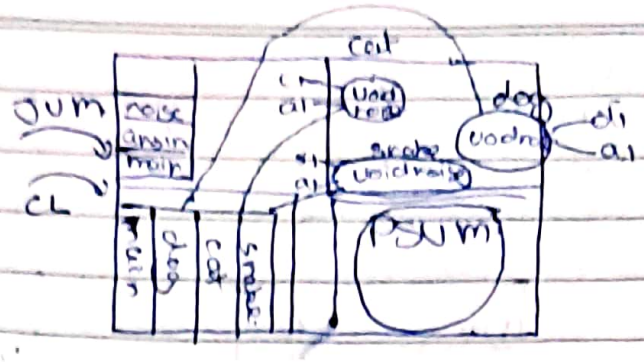
= consider a remote control it dosent matter if you are controlling a, tv, fan, or stereo - as long as they respond to common commands like. turn on & "increase volume" you can use remote to control any of them.

Polymorphism lets you interact with different types of object using a single point of contact.




```
Class animal
{
void noise()
 {
  sop(some noise);
 }
} class cat extends Animal
 {
  Void noise()
  {
   sop("meow meow");
  }
} class dog extends Animal
 {
  void noise()
  {
   sop("bow bow")
  }
} class snake extends Animal
 {
  void noise()
  { sop("buss buss");
  }
}
```

```
Class stimulator{
Static void ansim(animal a)
      a1.noise()
 }
}
Public class mainclass{
PSum(str[]args){
 Cat C1 = new cat();
 Dog D1 = new Dog();
 Snake S1 = new snake();
Stimulator.ansim(C1);
Stimulator.ansim(D1);
Stimulator.ansim(S1);
}
}
```

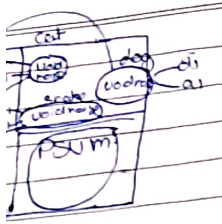


instance of operator - It is an operator it checks the given object belongs to the class heirarchy or not, if it belongs it will return true else it will return false.

```
Class whatsapp.
{
 Void send (int no)
 {
   Sop("sending no")
 } void send(str text)
   {
     sop ("send text");
   }
 Void send (int no, str text)
   {
     sop ("sending text & no");
   }
 Void send (string text, int no){
       sop("sending text&no");
   }
}
```

```
Static void Hg (animal a1) =null
        if (a1 instance of Animal){
            a1.noise();
        }
```

for understanding what is compile time polymorphism & static binding.



- It is an operator it checks
belongs to the class hierarchy
ngs it will return true else

```
)
no")
text)
xt");
, str text)
g text&no");
g text, int no){
text&no");
```

```
a1) = null
f Animal){
```

le time polymorphism &
nding.


```
class mainclass
{
psum(Str[] args)
{
whatsapp w1= new whatsapp();
    w1.send("Hi", 26);
    w1.send(true)
    w1.send(26)
    w1.send(61, "ok")
```

Javac mainclass.
error
Java.

for understanding
OOP

Static binding we cannot handle at compile time. ~~as it compile time as it will give error~~ RTP is dyramic as we can handle it as if we pass null that in compile time it wont give an error as we can pass null in Reference variable at Runtime it will give error. because we use instance of & null is not an object.

for OOP

of animal hierarchy consider the case of method overloading for this.

Abstraction:

Hiding the complexity of the system & exposing only the essential or 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 subclass. create a reference variable of interface type and initialize that reference variable with the implementation class object. this is how we achieve abstraction.

Scanned with OKEN Scanner

through abstract class we can achieve upto 100% abstraction. through interface. we can achieve 100 % abstraction.

Note- when to go for abstract class & interface

- when we dont know 100% implementation then we will go for interface.
- when we know partial implementation them we will go for abstract class.

for understanding

We will declare essential features in an interface. like 2-3years experienced developers use this if the want to decentralize the task to different teams he uses interface so if any of the method or feature is not overridden the class will become abstract and object cannot be created for it so software won't execute.

We can create reference variable for interface

HW write 5 programs to demonstrate abstraction.

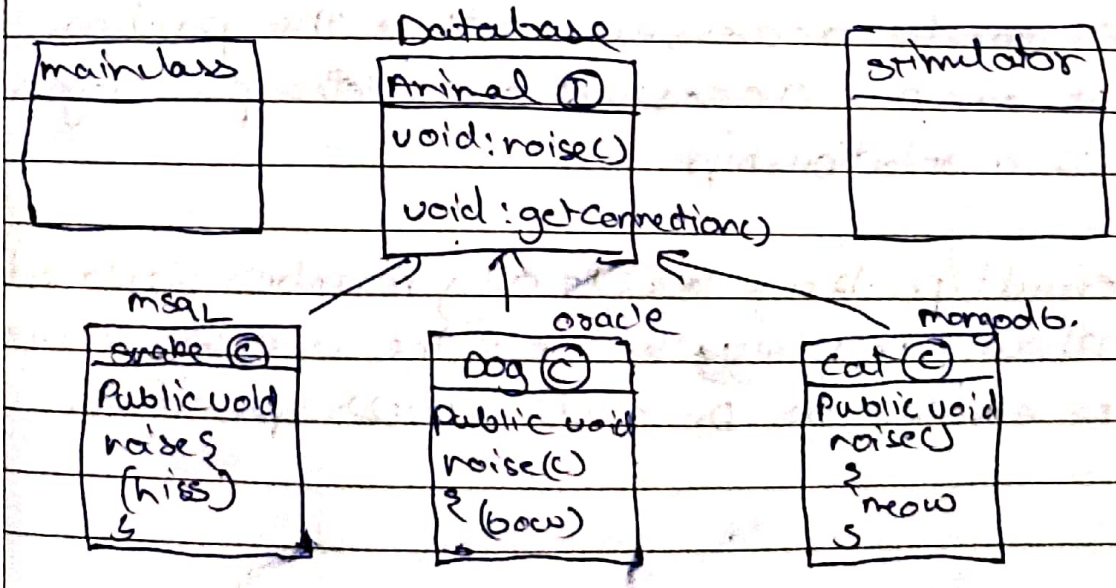
```
interface Animal
  {
   void noise();
  }
class cat implements Animal
  {
Public void noise(){
      sop("meow meow")
  }
class dog implements Animal.
  {
    Public void noise() {
      sop ("bow bow");
  }
```

papergrid

Date: / /

```
class snake implements Animal
{
  void noise() {
    Sop ~~sod~~("buss buss");
  }
} class stimulator
  static void ansim (Animal ai)
    {
      ai.noise();
    }
  } class mainclass
  { psvm (str [] args) {
    cat c1 = new cat();
    dog d1 = new dog();
    snake s1 = new snake();
    stimulator.ansim(c1);
    Stimulator. ansim(D1);
    Stimulator. ansim(S1);
  }
}
```

Abstraction- through abstraction we can achieve user friendly.
security



Access specifier.

It is used to restrict the access from one class to another class or from one package to another package.

In access specifier we have four types.

1) Public
2) Protected
3) default/~~pac~~ package level
4) Private

Public- any member declared with the keyword public is called as public access specifier.

- It can be accessed
1) within the class
2) It can be accessed within the package.
3) It can be accessed ~~within~~ outside the package.
4) It can be accessed anywhere.

Protected- any member declared with the keyword protected is called as protected access specifier-

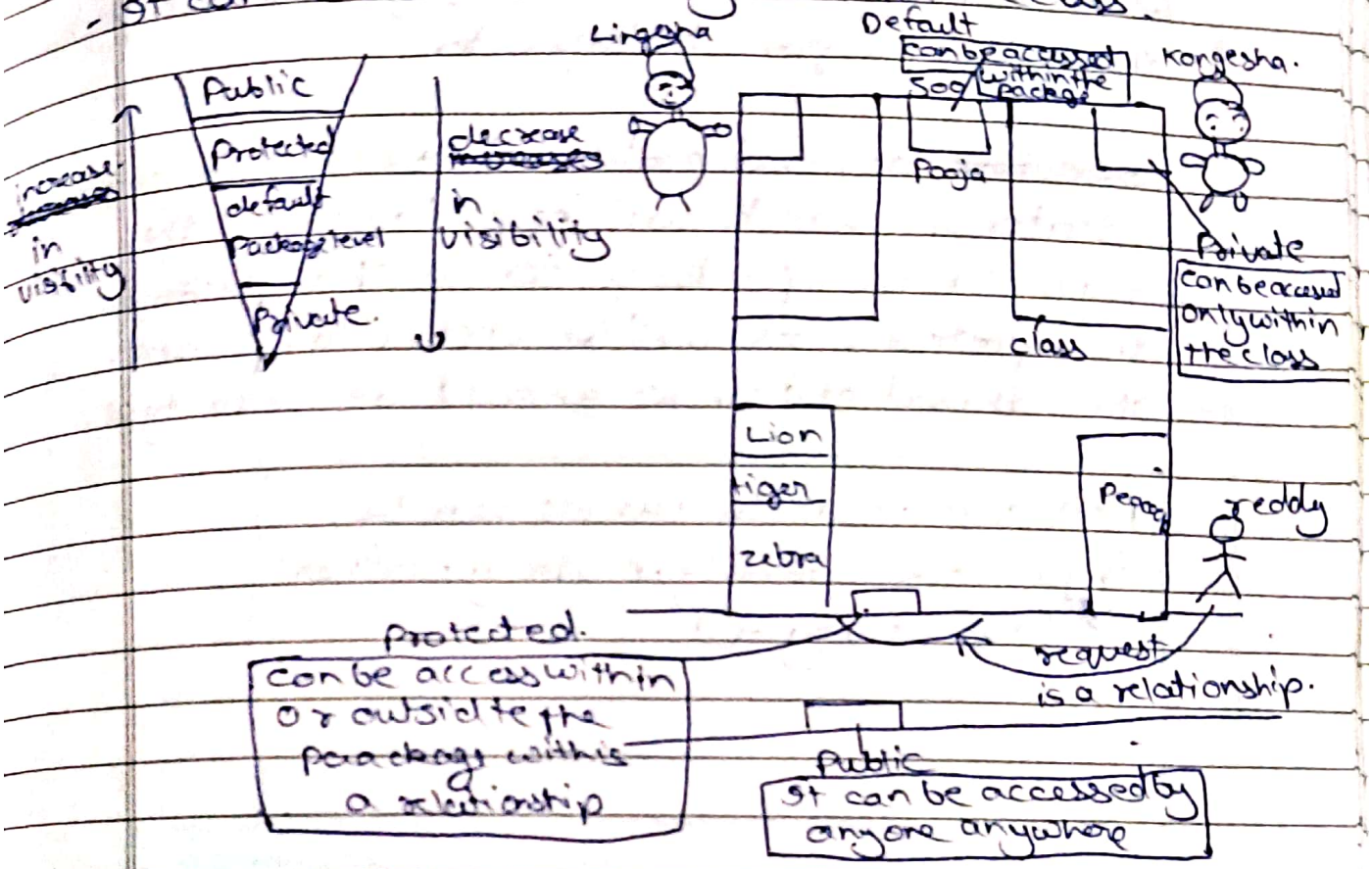
1) It can be accessed within the class.
2) It can be accessed within the package
3) It can be accessed outside the package with is a relationship

3) **Default/package level** = any member declared without any keyword for the access specifier is called as Default access specifier.

Scanned with OKEN Scanner

- It can be accessed within the class
- It can be accessed within the package.

① **Private** - Any member declared with the keyword private is called as private access specifier.
- It can be accessed only within the class.



Note: ① a class can have 2 access specifier
Public & default.
② the members of the class can have all 4 access specifier.

```
Public
default.
        Class Sample
        {
Public      int a = 10
Protected       void disp()
default         {
Private         }
        void Sample()
        {
        }
```

in increasing order of visibility we can change access specifier from
private → default → protected/public → public.

Note - In a Java file the first statement should be the package statement, it says to which package the below program belongs to.

- ~~selection~~ second statement should be import statement which will help to import the files from 1 package to another package.
- the imported files will be virtually present.
- the third statement should be Java type.

```
Pkg com.family.uncle family;
import com.family.myfaily.father;
class aunty -③
{

}
```

Com
|
family

uncle family

```
class aunty extends father.
{

}.
```

myfamily

```
class father
{
Private void add()
void car()
Protected void bike()
Public void cycle();
}
```

Scanned with OKEN Scanner

papergrid

Date / /

```
Package com.family.myfamily
public class father{
    Private void atm(){
        Sop ("dad ka Atm");
    } void car{
        Sop ("dad ka car");
    }
    protected void bike(){
        Sop ("dad ka bikeuuu")
    } public void cycle(){
        Sop ("dad ka cycleuuu");
    } psum(str[] args)
    {
    father f1 = new father()
        f1.atm();
        f1.car();
        f1.bike();
        f1.cycle()
    }
}
```

~~Public class son~~

```
Package com.family.myfamily
  Public class son{
    Psum(str[] args)
    {
    father f1 = new father();
        f1.car();
        f1.bike();
        f1.cycle();
    }
}
```

Scanned with OKEN Scanner

```
Package com.family.unclefamily
import com.family.myfamily.father
    ~~psvm(str[] ar~~
class aunty extends father{
    psvm(str[] args){
        ~~father fr = new~~
    aunty a1 = new aunty()
        a1.cycle()
        a1.bike()
    }
}
```

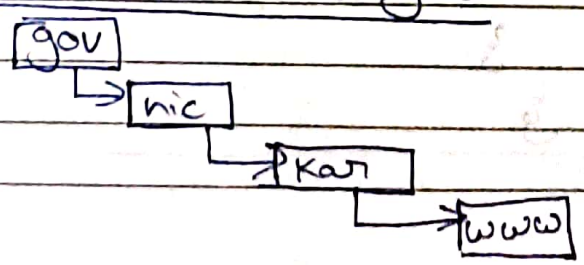
(com in place of father (*) this all is imported from package)

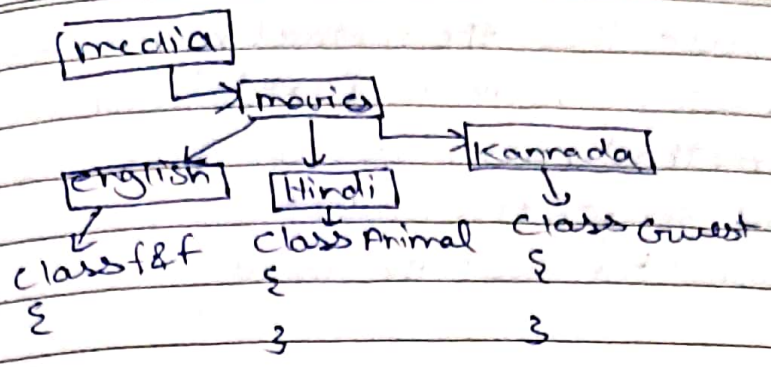
Package

1) Package are the folder structure which is used to store similar kind of data
2) package statements should be the first statement in any Java file
3) Java provide a keyword called import to import the files from one package to another package.
4) We can have n no of import statements.

- the third statement can be any Java type
- we can have multiple classes in a file whichever class is public it should have the main method & filename.
- whenever we create a folder structure it should be in reverse order of url.

URL = www.kar.nic.gov






```
package media.movies.Kannada; -①
import.media.movies.englishfuc;-②
import.media.movies.Hindi.Animal;

Class Ghost — ③
{

}
Public class Demo — Demo.Java.
{ psum(str [] args)
  {
  }
}.
```

Encapsulation

It is one of the OOPS principle in Java.
Java is by default encapsulated.
We cannot declare any variable outside the class
We cannot have any print statements outside the methods.
Declare the Datamembers as private & restrict the direct access outside the class & provide the indirect access through public services called. getters & setters is called as encapsulation.

Scanned with OKEN Scanner

- there is no rule that the method name should be get & set, it is an industrial convention to use the method name as get & se.
- get is used to get the value.
- set is used to set the value.

Java Bean class

Declare the data member as private & restrict the direct access outside the class & provide the indirect access through public services. called getters & setters is called as Java Bean class.

Q Write the program to add two no's with the dynam input. by taking the input byte, int, short, long, float, double.

Scanner class

It is a inbuit class which belongs to Java util package. which should be imported explicitly.

- It has the nonstatic methods which helps to take dynamic input from the users.

```
class scanner
{
 byte nextbyte()
 {
 }
 Short nextshort()
 {
 }
 int nextint()
 {
 }
```

Scanned with OKEN Scanner

long nextlong()
{
}
Double nextDouble()
{
}
char next().charAt(0)
{
}
boolean nextBoolean()
{
}
String next() {
}
String ~~nextline~~ nextLine()
{
}

next()- reads ~~next token~~ until the first whitespace
nextline()- reads until the end of line.

exp- scanner sc = new scanner(system.in);
String a = ~~next~~ sc.next();
(user enters "Hello world")
- It will take Just Hello.

exp- scanner sc = new scanner(system.in);
string a = sc.nextline();
(user enters "Hello world")
- It will take Hello world.

Scanned with OKEN Scanner

papergrid
Date / /

```
Package PGM;
import Java.util.scanner;
Public class sample1{
    Static void dynamicInput()
    {
    Scanner sc = new Scanner (System.in);
        System.out.println("enter the no");
        int no = sc.nextInt();
        S.op(no + no);
    }
    static void add()
    {
    Scanner sc = new scanner(System.in);
     Sop("enter the 1st no")
     int no1 = sc.nextInt();
     Sop("enter the second no");
     int no2 = sc.nextInt();
     Sop(no1 + no2);
     }
     PSVm(str[] args)
       add();
       dynamicInput();
     }.
}.
```

HW
Q Write 5 programms with realtime exampleon encapsulation.

```
Class ICICI Bank
{
Private int password = 1234;
  public int get pwd()
  {
    return password;
  }
Public void set pwd(int password).
    {
      this.passwword = passwword;
    }
  }
class mainclass
{
P.SVM (Str[] args)
  {
  ICICI Bank atmcard = new ICICI Bank()
        int x = atmcard.getpwd();
          Sop(x)
      atmcard.setpassword(3456);
        Sop.(atmcard.getpassword());
      }
    }.
```

atmcard → Private int pwd
public int get pwd
Public void set pwd(int pwd)

Note- we can directly print the method or use method in print statement if return type is anything rather than void.

Scanned with OKEN Scanner

```
Package encap;
class phonepe
{
Private string pwd= "alinga";
 Public string getpwd(){
        return pwd;
  }
Public void setPwd(string pwd){
        this.pwd = pwd;
  }
} public class mainclass1{
    Psvm (Str[] args){
        Phonepe P1= new phonepe();
          sop(P1.getpwd());
         P1.Setpwd("mangi");
          sop(P1.get pwd());
    }
}.
```

Exception Handling-

It is an event triggered during the execution of the program which interrupts the execution and stops the execution abruptly or suddenly handling this event by using try and catch or throws is called as exception handling.

- all the abnormal statements should be developed in the try ~~catch~~ block and address them in the catch block.
- we cannot develop any statement between try & catch block.

Scanned with OKEN Scanner

papergrid
Date: / /

- for one try there should be minimum one catch block.

- we cannot develop try block alone it will throw CTE
* exception can be handled in 2 ways.
1) try & catch.
2) throws.
- for one try block we can have more than one catch block
- all the exception can be addressed by Exception class.
- all the exception belongs to Java.lang. package.

exception's are of 2 types.

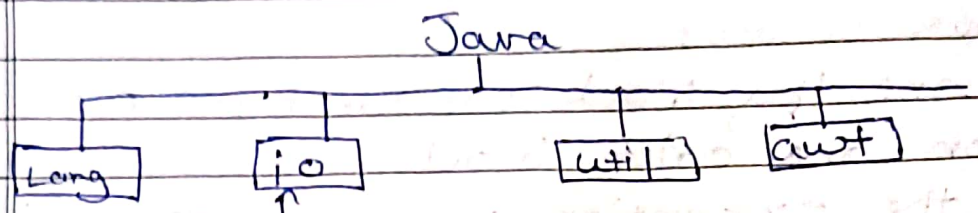
1) compile time exception / checked exception.
* any exception which is occurred at the compile time is called as compile time exception.
- compile time exception can be handled using try catch & throws.
- its should be addressed at the compile time itself.

2) Run time exception / unchecked exception -
* any exception which is occured at the Runtime is called as Runtime exception.
- Runtime exception can be handled using try catch & throws.
- it should be addressed at Runtime itself.
- once the exception is addressed we cannot readdress it will through CTE.

Scanned with OKEN Scanner

Object class

Java library - Java libraries are the inbuilt folder structure which consists of classes & interfaces.
we have inbuilt packages as follows.




```
class object
{

}
class string
{

}
Class Exception.
{

}.
```

```
classfile
{

}
```

- Object class is the supermost class in Java, each and every class extends object class.
- object class will be imported to each & every package.
- It has following methods.
  int hashcode()
  boolean equals()
  object clone() used to copy object.
  String tostring()

going to study in threads

void notify()
void notifyAll()
void wait()
void finalize()

D) toString()

- to String() is a non static non final method of object class.
- each & every class inherits to string methods.
- whenever we print the reference variable to String method gets invoked where it will return the object address.
- the signature of two string method is public String toString().
- we can override toString method.

```
package demo
public class sample extends object
{
    PSVM(Str [] args){
        sample S1= new sample();
        SO (S1. to string());
    }
}
```

O/P = Demo.Sample @ 273014d.

Steps to override toString()-

Step 1 - consider the attributes
Step 2 - initalize the attributes using constructor
Step 3 = override to string method.

Q employee id, employee salary & employee name. Wap using to string method overside.

Scanned with OKEN Scanner

papergrid

Date: / /

```
Class employee
{
char employee_id;
double employee_sal;
string employee_name;

employee(char employee_id, double employee_sal; string
            employee_name)
{
        this.empid= empid ;
        this. empsal= empsal;
        this. empname= empname;
}.
String to string()
{

        return this. empid+" "+ this.empsal+" "+
                                this. empname
}.
psum(str[] args)
{
  employee E1= new Employee(A, 123, "Ray")
      sop(E1);
}
}
```

HW Write 10 programs to over ride to string method.

Q for student, sec, & std name

papergrid
Date / /

not needed.

```
Class Student extend object.
{
char stdsec;
string stdname;
int stdid;
Student(char stdsec, string stdname, int stdid){
  this.stdsec = stdsec;
  this.stdname = stdname;
  this.stdid = stdid;
}
Public string tostring()
{

  return this.stdname+" " this.stdsec+" "this. std id;
}
  psvm (str [] args)
  {
  Student S1 = new student ('A', "Dinga", 123)
        sop (S1. to string());
    }
  }.
```

```
Char stdsec = A
String stdname = Dinga
int stdid = 123
P string to string
{
return(this.stname+" "+this.stdsec+"
  + this. std id);
}
```

S1

Scanned with OKEN Scanner

- equals method()
- It is a non static non final method of object class.
- each & every class ~~inherits~~ extends object class
- equals method will compare two object addresses if both the address are same it will return true else it will return false.
- the signature of equals method is public boolean equal(object obj)
- We can override equals method

```
Package obj
public class sample
{
    Public static void main(String[] args)
        sample s1 = new sample();
        sample s2 = new sample();
        sop(s1.equals(s2));
    }
}
```

Steps to override equals method.

1. consider the attributes
2. Initialise the attributes using constructor while creating object
3. override equals method.
   - upcasting
   - downcasting
   - comparison

Q car name, car color, car price compare the car price.

Scanned with OKEN Scanner

papergrid
Date / /

```
class car extend object
{
  str carname;
  str carcolor;
int ~~str~~ carprize;
  car (str carname, str carcolor, int carprize)
  {
    this.carname = carname,
    this.carcolor = carcolor,
    this.car prize = carprize;
  }
  boolean equals (object obj)
  {
   car e2 = car(obj);
  return this.carprice = e2.carprize;
  }
  psvm(str[] args)
  car c1 = new car("sedan", "Red", 12345);
  car c2 = new car("hatchback", "Blue", 12347);
     if (c1.equals (c2))
          Sop ("price is same");
       else {
          Sop ("different price")
          }.
```

WAP 10 programs to override equals method.

```
Class Employee
 {
  int empid;
  double emp sal;
  string emp name;
```

Scanned with OKEN Scanner

```
Employee (int empid, double empsal, String emprame
{
    this.empid= empid;
    this.emp sal= empsal;
    this.emprame= emprame;
}
Public boolean equals (object obj)
{
    employee e2= (employee) obj
  return this.emprame == emprame;
}
Public static void main (Str[] args)
{
  employee e1= new employee (112, 3000.56, "Ringa")
  employee e2= new employee (124, 30000.56, "Ring

  if (e1.equals(e2)){
          sop ("emp name is same");
  } else{
        sop("emp name is different");
      }
  }
}.
```

Hashcode method

- It is a nonstatic nonfinal method of ~~Sout~~ object class
- each & every class inherits Hashcode methods
- whenever we ~~have~~ invoke the hashcode method it will generate a unique integer no called hash no. based on the object address.
- the signature of hashcode method is public int hashcode.

Scanned with OKEN Scanner

```
Public class demo1{
psvm (Str [] args){
Demo D1= new demo1();
sop (d1. hashcode());
Demo1 d2 = new demo1();
sop (d2. hashcode());
}
}
```

O/p- 36671642
1829164700

for understanding

* hashcode() method in Java is like a unique Id ~~for~~ assigned to an object. It is used for efficient storage & retrival in certain data structures. if two objects are equal then there hascode should be equal too. It is a way for objects to be quickly organized in collections like set & maps making operations like searching & sorting faster.

```
class sample
{
Public int hashcode()
{
  return 123;
}
psvm (Str[] args){
    sample s1 = new sample();
        sop (s1. hashcode());
  }
}
```

Scanned with OKEN Scanner

papergrid

Date / /

Final class

- any class whic is declared with the keyword final is called as final class.
- we can declare variable & methods as final but not the constructor
- we can create object for final class but we cannot inherit final class
- we can inherit final method but we cannot override.
- we can overload final method

①
```
final class sample{
  final int a=10;
  final void disp(){

  }
  final sample() ✗ we cannot create constructor
  {                as final
  }

class demo ~~extends~~ sample ✗ we cannot inherit
                               final class.
  {

  }
```

②
```
class sample1
{
final void disp()
  {
  sop("hello");
  }
class test extens sample1 {✗ we can inherit final
  final void disp() method but we cannot overrid
  {
    Sop ("hi");
  }
```

3) class tester
{
final void disp(double y)
{

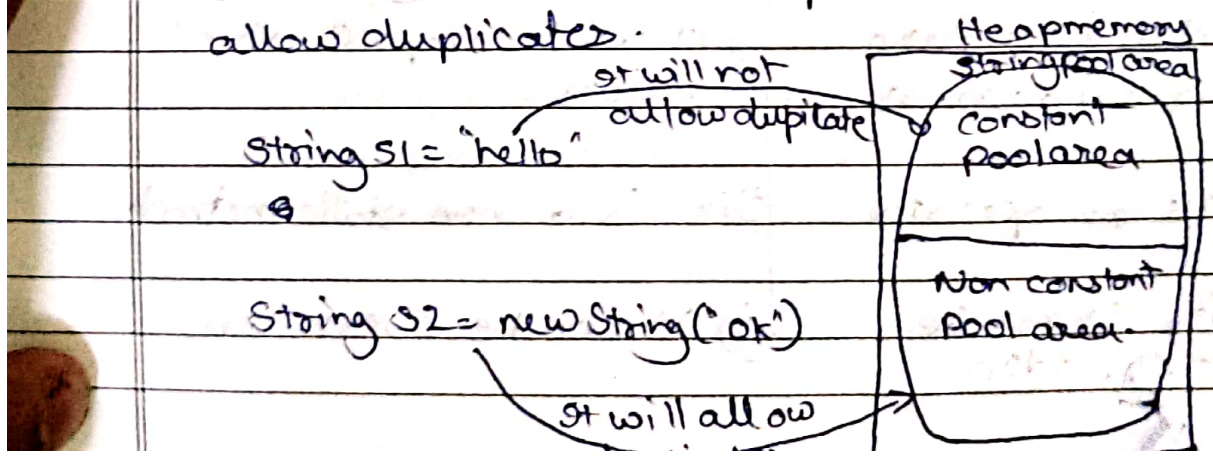
}
final void disp(int x)
{

}

we can overload final method.

String class

- string class is a final class which belong to Java.Lang package.
- we can create string object in two ways
1) string object without new operator
2) string object with new operator.
- All the string object will get stored in string pool area under heap memory.
- string pool area is divided into two parts
1) constant pool area, which will not allow duplicates
2) non constant pool area which will allow duplicates
- string object which is created without new operator will get stored in constant pool area, which will not allow duplicates.
- string object created with new operator will get stored in non constant pool area which will allow duplicates.

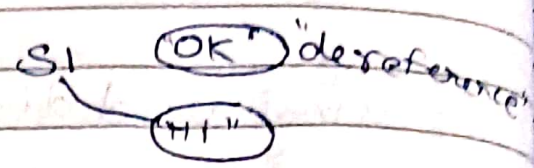


- String class is immutable.

What?

immutable means it will not change the state of the object.

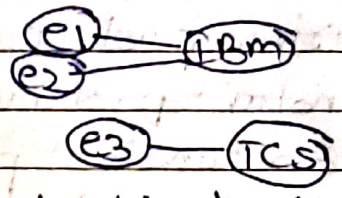
String S1 = "OK"
S1 = "Hi"



why?

when multiple reference variable as pointing to a single object and if one of the referance variable is reninitalized it will not affect the other referance variable this is why it is immutable.

String e1 = "ibm"
String e2 = "ibm"
String e3 = "ibm"
e3 = "TCS"



Proff that with new keyword object allows duplicate without new keyword will not allow

==	equals()
Comparison operator	wrt. String
1) It will compare object address	1) it will compare object values
2) it is an operator	2) it is a non static method.
3) it cannot be overridden	3) It can be overridden

Scanned with OKEN Scanner

```
Public class demo {
 psvm (str [] args)
 {
 String s1 = "ok";
  String s2 = "ok";
 sop( s1 = = s2);
 string s3 = new String ("hi");
 string s4 = new string ("hi");
 sop (s3 = = s4);
 }
}
```

o/p = true
False.

```
Public class hello {
    psvm (str[] args){
        String s1 = "hello";
        String s2 = new string ("hello");
        sop(s1 = = s2);
        sop( s1.equals(s2));
    }
```

o/p = ~~true~~ false
true

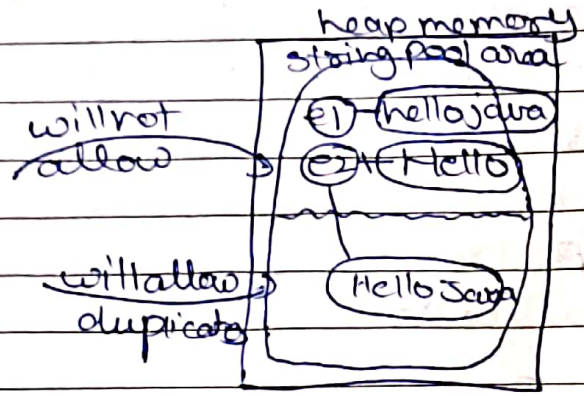
equals method is overridden in such a way that it compares object values.

Note- any operations performed on a string object the result will get stored in non constant pool area.

Scanned with OKEN Scanner

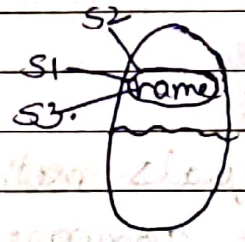
```
String e1="Hello Java";
String e2 = "Hello"
e2= e2+"Java";
//Hello + Java
sop(e1 == e2);
```

O/p= false



Java is Robust because of proper memory management like in case of string class if we create multiple string object with the same value then a single object will be created and all reference variable will be Pointing towards that object

```
String s1= name
String s2= name
String s3= name
```



Exception Handling-

It is an event triggered during execution of program which interrupts execution or stops the execution abruptly or suddenly handling this event through try catch & throws is called as exception handling.

Scanned with OKEN Scanner

Syntax for try & catch:-

① try {
}catch(exception class refvariable)
{
}

② try {
}catch( ){
}catch( ){
}

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

④ try {
}catch( ){
try {
}catch( ){
}
}

Scanned with OKEN Scanner

5) try{
}finally{
}

6) try{
}catch( ){
}finally{
}

- arabian fish
  amazon
- helpbutton on a page
- Phone pe checkbalance
- Road hump

```
① Public class sample1
  {. psvm( str[] args){
    sop("* * * * * * main starts * *")
    try {
      int i = 1/0; //throw new Arithmatic exception
    } catch (Arithmatic exception e) ("/zero")
      {
        sop ("Handled");
      }
    sop("* * * * * * main ends * * **");
  }.
}.
```

throw will create an object for the following exception class.

Scanned with OKEN Scanner

```
(2) class Demo {
    psum (str [] args) {
      sop("* * * main starts * * *");
        int[] arr = {10, 20};
        try {
       sop(arr[97]);
        } catch (ArrayIndexoutofboundexception e)
        {
            sop("caught");
        } sop("* * * main ends * * *");
        }
        }.
(3) public class Sample2 {
      psum (string [] args){
          sop("* * * * main starts * * *");
          sample2 s1 = null;
          try{
          sop (s1.hashcode());
          }
          catch(NullPointer Exceeption e){
                sop("handled");
          } sop(* * * * main ends * * * *);
        }
        }
(1) class sample3 {
    psum(str[] args){
      sop("main starts * * *");
      try{
          int a = Integer.parseInt("sdsd")
                sop(a);
          }catch (numberformat exceptione){
                sop("handled")
          } sop(* * * main ends * * * *);
```

Q WAP to demonstrate try catch catch.

```
PSVM(str [] args){
    Sop(mainstarts* * *);
    try {
        int i = i/0;
    } catch(ArrayIndexoutofBoundsexceptione){
            sop("caught")
    } catch (Airthmatic exceptionC){
            sop ("Handled");
    } sop("mainends");
    }
}.
```

o/p-mainstarts
handled
mainends.

Q WAP to demonstrate nested try catch
- We cannot develop any statements between try & catch block
- once the exception occurs further statements of the try block will not get executed

```
PSVM (str [] args)
  { sop ("main ~~starts~~");
  try { int i = 40/2;
      int [] arr = {10, 30, 20, 50, 60, 80};
      try{
          sop (arr[i]);
      } catch (array index out of boundexception
          sop("caught")
      }
  } catch (Arithmatic exception e) {
          sop("handled");
  } sop("main ends");
}
```

Q- when there is code dependency we go for nested try catch

exp- from an atm your will withdraw 500Rs but in your account you have 199 then in that case there will be use of nested try catch.

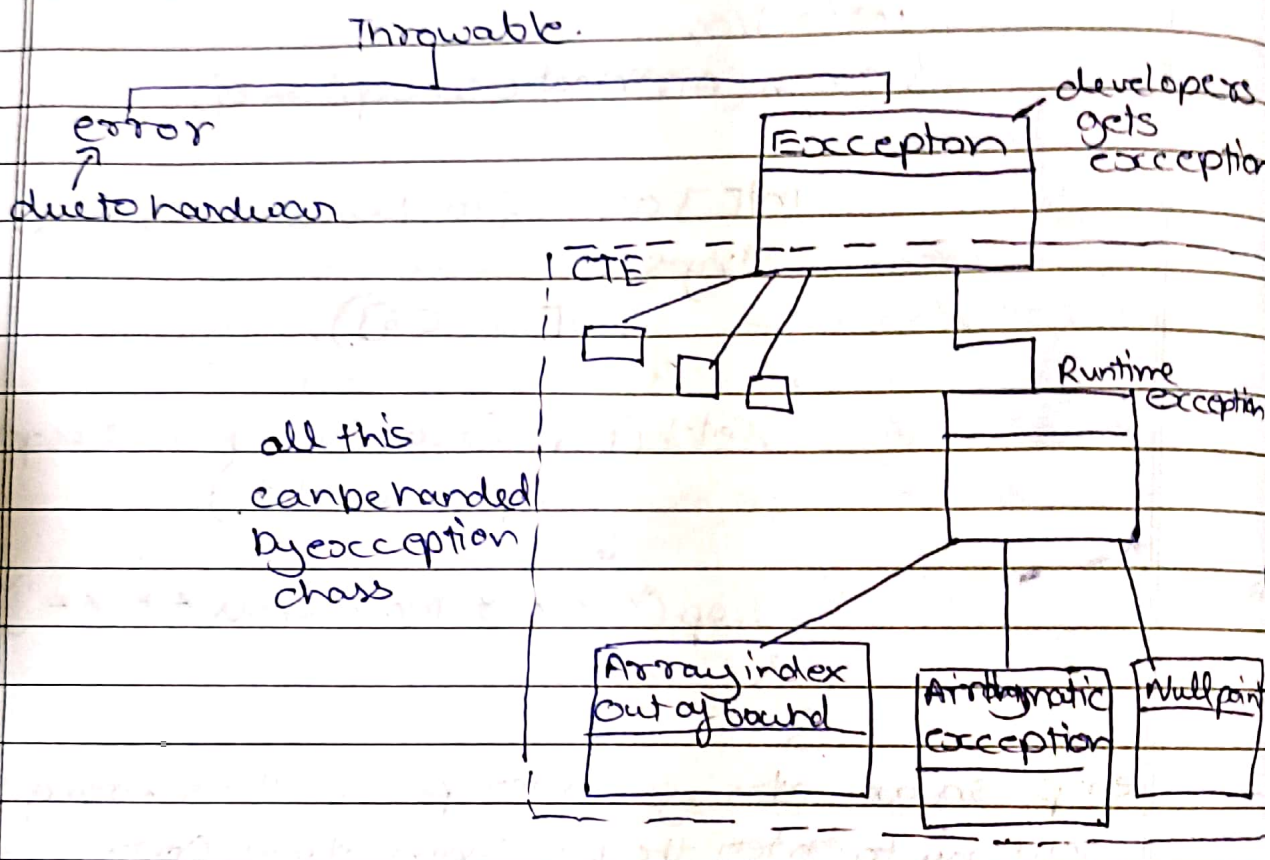
Q- WAP to demonstrate try catch in a catch block.

```
- psvm(str[] args){
    sop("* * * * main starts * * * *");
    try{
        int i = 1/0;
    } catch(ArithmaticException e)
        {
            int[] arr = {10, 20, 30, 40, 50};
            try {
                sop(arr[8]);
            }
            catch(ArrayIndexOutofBoundException e){
                sop("caught");
            }
            sop("* * * * main ends * * * *");
        }
    }
```

exp- on an atm you enter pin 1234 its wrong so it tells you to enter the pin again it will go to catch block & display message its wrong & reattempt. so if you enter 3 time wrong pin then it will display message account is blocked.

Scanned with OKEN Scanner

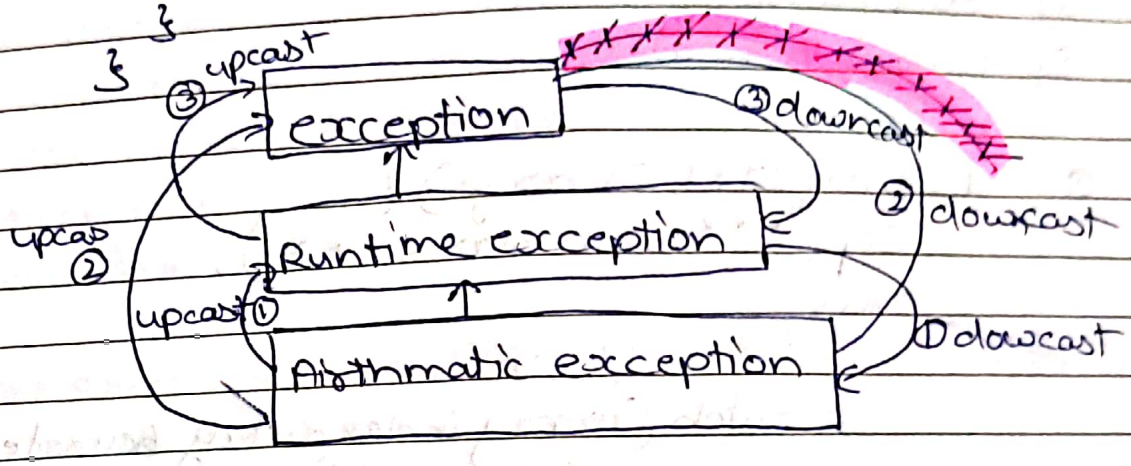
```
try {
  Pin
} catch {
    reattempt 3 times / wrond attempts.
        count = count + 1
    try { count == 3
        } catch {
            account is blocked.;
        }
    }.
```



"Throwable" is a base class in Java. for all exceptions & error

Note: once the exception is addressed if we try to readdress the we will get compile time error.

```
psvm (str[] args){
  sop("mainstarts");
try{
  int i = 1/10;
}catch(Airthmatic exception E1)
  {
    sop("handled");
  }catch(Exception E)
  {
    sop("caught");
  }sop("mainends");
  }
}
```



① when the catch block will get executed?
whenever there is an object created for the exception class.

② In the try block after error the remaining statements are not executed. why?
because if there is an exception in the try block then the object of that exception will be created. & the control will move to the catch block & it will get executed & the control cannot traverse back as java executes from top to bottom so nie ther the further statements in the try block will get executed nor the statement between try & catch will get executed

- we cannot create statements between try and catch as compiler will get confused that what to execute so it will give compile time error

Finally block

Finally block is a block in exception handling which will get executed mandatorily eventhough the exception is addressed or not addressed.

Finally block can be developed in two combinations
1) try-finally
2). try-catch finally.

```
@ PSVM(Str[] args){
    Sop("*** main starts ***");
    try{
        int i = 1/0;
    } catch(ArrayIndexOutOfBoundException e){
        Sop("handled");
    } finally{
        Sop("I am finally block")
    } Sop("main ends***").
  }
}
```

we cannot place a finally block between try and catch as after an exception in try block it will search either for a catch block or for a finally block. so if finally is in between then there will be confusion & compiler will throw compile time error.

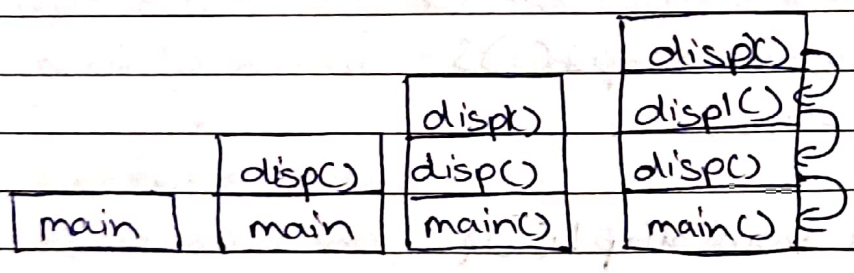
Scanned with OKEN Scanner

2)
```
psvm(str[] args){
  sop("* * * * main starts * * * *")
  try {
    int i = 1/0;
  } finally {
    sop("I am a finally block");
  } sop("* * * main ends * * *");
}
}.
```

Stack unwinding

If an exception occurs in any one ~~class~~ method it will propagate to the called methods intern to the main method if there is no handle the stack will get destroyed here it is called stack unwinding.

Print stacktrace() method - It is a nonstatic method of throwable class which is inherited to each & every class it prints the complete ~~stack~~ stack trace which is stored in system.err.



"if not handled then complete stack will get destroyed"

papergrid

Date: / /

```
class throwable
{
void printstack trace( )
 {
 }
 } class exception extends throwable
  {
  }
  class Runtime exception extends throwable
  {
  }
  class Airthmatic exception extend Runtimeexcep
  {
     void print stack trace ( )
     {
     }
  }
}
                class System
                {
                Static string
                ~~System~~ . err = " ";
                }

Public class demo1
{
  Static void disp2( ){
   int i = 1/0;
  }
  Static void disp1( ){
     ~~int~~ . disp2( );
  }
  Static void disp( ){
     disp( );
  }
```

```
psvm(Str[] args)
{   sop("main starts");
    try {
        disp();
    }
    catch (Airthmatic exception e){
        ~~sop("main ends");~~ e.printStackTrace();
    } sop("main ends");
}
```

O/P
```
main starts
Java.lang.AirthmaticException:/zero
    at ~~ex~~ packagename.classname.methodname(line no)
                    "
                    "
    at packagename.classname.methodname(line no)
main ends
```

Throws:-

Throws keyword is used in the method declaration ~~to indicate that method might through an~~ to throw the specified exception to the caller method. & signals the caller method that they need to handle ~~or~~ propagate these exception.

checked exception

```
class shadi.com{
 static void submit( ) throws marriage exception
  {
      int age=14;
        if(age >=21){
          sop("happy life");
        } else {
            throw new Marriage Exception("Invalid age")
```

```
psvm(str[]args)
{ try{
      submit();
    }
    catch (marriage exceptione){
    }
}
class marriage exceptionextendsexception
{
private string msg;
    marriage exception(string msg){
        this. msg = msg;
    }
    Public string getmsg()
    {
        return MSG;
    }
}.
```

For understanding

why we do not print simple message insted we go for exception handling-

- we go for exception creationto standardise the o/p message for the exception occurred. like in shadi. com if there is a age exception in login, home & apply pe & we printed o/p message as Invalid age, age not valid wrong age. so to standardise it as a invalid age we go for exception handing. we use throw in all the 3programs login, home & apply in order to get a standardise message.

```
class AmazonException extends exception.
{
 String msg;
 AmazonException(String msg){
     this. msg = msg;
  }
 Public string getmsg( ){
     return MSG;
  }
 Class Amazone{
  static void order( ) throws AmazonException{
     int price = 10000;
     if (Price < 5000){
        sop("Its a good discount")
        else{
          throw new AmazonException("no discount");
       }
  } PSVM (str[] args){
     try{
         order();
       }
       catch(AmazonException e){
       sop (e. get msg());
       }
     }
```

No. Write 5 programs on checked exception.

File Handling

File is nothing but the similar kind of data & performing operations on a file like creating, deleting & modifying the file is called as file handling.

Scanned with OKEN Scanner

- File class belongs to Java.IO. package which should be imported explicitly.

- It has following methods-
  Create the file - boolean createNewfile(){}
  Create the folder - boolean Mkdirs(){}
  delete - boolean delete(){}
  exists - boolean exists(){}

Q - Write a program to create a folder to check weather it exists or not & delete the folder.

```
import Java.IO.File;
class sample{
   Psvm(Str[] args){
File F1= newFile("C:\\ user\\prabh\\ onedrive\\ Desktop\\ hello\\ hi");
 if(f1.mkdirs()){
   Sop("folder created");
 }else{
      Sop("folder not created");
 if(f1.exists()){
   sop("folder exists");
  } else {
 Sop("folder dosen't exists");
 } if(f1.delete()){
   sop("folder is deleted");
  } else {
    ("folder not deleted");
    }
}
}
```

Q- WAP to create a file.

Scanned with OKEN Scanner

```
import Java.io.File;
import Java.io.IOException;
  class Sample{
  psvm(str[]args){
File Fl = new File("C:\\user\\prabh\\ondrive\\Desktop\\hello\\prabhat.pdf")
  try{
      if (fl.createNewFile()){
          sop(file is created.
      }
    } catch(IOException e){
        e.printStacktrace();
      }
    }.
```

As the same address will not be in every file and Java is platform independent language that means (write once execute anywhere) so there will be an exception to handle it we use try & catch & import a class IO.Exception.

drawback of Java-

we cannot store data in case of dynamic input the data is not stored. everytime we run the program we will have to take new input so to overcome this we have Jdbc.

Q- WAP to write the data into the file.

File writer & Filereader.

- File writer & File reader are the inbuilt classes which belongs to Java.IO package.
- File writer class has write method to write the data into the file.
- File reader class has read method to read the data from the file.

Scanned with OKEN Scanner

```
Pkg Java.IO;
class filewriter
  {
   void write(){

  }
}
```

```
Pkg Java.IO;
class filereader
  {
   void reader (){
  }
}.
```

for understanding
in file class a method length() is there which is of long return type but in case of array we will typecast the value to integer to return integer value.

- we should always use flush() after writing the data.

→ to dump the data into the hardware we use flush().

S/W eclipse $\xrightarrow{\text{flush}}$ H/W

To write

```
import Java.io.File;
import Java.io.Filewriter;
import Java.io.~~File~~ IOexception;
Class sample{
 psvm (str[] args) throws IO Exception{
   File Fl = new File("C:\\user\\prabh\\onedrive\\Desktop\\
                        hello\\humara.txt");

    Fl.CreateNewFile();
   File writer FW = New File writer (Fl);
      Fw.write("hello java");
      Fw.write(" hello mava");
      Fw.write("hello bauxa");
      Fw.Flush();
```

Scanned with OKEN Scanner

```
Sop("Data is written")
}
}.
```

IO read.

```
import Java.io.File;
import Java.io.FileReader;
import Java.io.IOException;
class filereader{
psvm(str[]args) throws IOException{
File F1 = new File("C:\\users\\prabh\\onedrive\\Desktop\\hello
FileReader FR= new Filereader(F1);   hurrara.txt");
    char[]arr= new char[(int) f1.length()];
                FR.Read(arr);
        //  for (int i=0 ; i<arr.length; i++){
1st     //          sop(arr[i]);
        //      }
        another method.
2nd     //  String s1= new String(arr);
                sop(s1);
            }
        }.
```

Q- WAP to demonstrate buffered Reader & buffered writer.

- buffered reader & buffered writer are the inbuilt classes which belongs to Java.Io package which helps to improve the performance of writing & Reading the data.

for understanding

buffered data is easy to recall like I take Sname the name I took at last will be easy to recall than the first name this is how it works.

Scanned with OKEN Scanner

papergrid

Date: / /

- **bufferedwriter**

```
import Java.io.Bufferedwriter;
import Java.io.File;
import Java.io.Filewriter;
import Java.io.IoException;
Public class bufferedwriter{
  psvm(string[] args) throws IOExcepton {
    File Fl = new File("c:\\users\\prabh\\onedrive\\Desktop\\
                        hello\\prabhash.txt");
      Fl.createNewFile();
  FileWriter FW = new FilWriter(Fl);
  BufferedWriter BW = new Bufferedwriter(FW)
      BW.write("hello how Re you");
      BW.newline();
      BW.write("chalo jaun you");
      BW.newLine();
      BW.write(" Bonjour");
      BW.newline();
      BW.Flush();
    }
  }
```

- **Buffered reader -**

```
import Java.io.Bufferedreader;
import Java.io.File;
import Java.io.Filewriter;
import Java.io.IoException;
Public class buffered Reader{
psvm(str[] args) throws io exception {
File Fl = new File("c:\\ users\\ prabh\\ onedrive\\ hello\\
                    Prabhash.txt");
FileReader FR = new File Reader(Fl);
BufferedReader BR = new BufferedReader(FR);
```

Scanned with OKEN Scanner

```
String S1 = BR.readLine();
while(S1!=null){
    sop(S1);
    S1= BR.ReadLine();
    }
}
}
```

- readline()- its a method through which we can read the content line by line directly;

File inputStream & File output stream

these are the inbuilt classes which belongs to Java.IO package. hence to read & write the media files.

2 write a program to read & write an image-

```
import Java.io.File
import Java.io.FileInputStream;
import Java.io.FileOutputStream;
import Java.io.IOException;
public class inputoutputstream{
psvm(str[]args) throws IOException{
    File Fl= new File("C:\\users\\prabh\\onedrive\\desktop\\
                image-wallpaper.jpg");
    FileInputStream FIN= new FileInput Stream(Fl).
        byte[]arr= new byte[(int) Fl.length()];
        FIN.Read(arr);
    FileOutput Stream Fout= new File Outputstream("C:\\
                users\\prabh\\onedrive\\hello.jpg");
        Fout.write(arr);
        Fout.Flush();
    }
}
```

Threads:

Thread is an execution instance which owns its own CPU time & memory is called as thread.

- thread is an inbuilt class which belongs to Java.lang package which is imported to each & every package.

When to use try & catch - use this when you want to deal with problem right where it happend. it is mainly used to handle exception within the method.

Throws - use it when you do know how to handle the problem. its like telling the method. that I am not handling the problem here. the code using me. needs to handle it.

Sleep() - its a static method of thread class.

```
package Java.lang;
    class thread {
        static void sleep(int millisec){
        }
    }
```

exp -

```
import Java.lang
    class Demo12
    {
    psvm(Str[] args){ Throws InterruptedException
    for(int i=0; i<=10; i++){
        sop(i);
        threads.sleep(1000);
    }
    }
```

- Sleep method takes input in milliseconds.

Scanned with OKEN Scanner

windows -> showview -> Package explorer.

papergrid
Date: / /

- Sleep method has its own Interrupted exception.

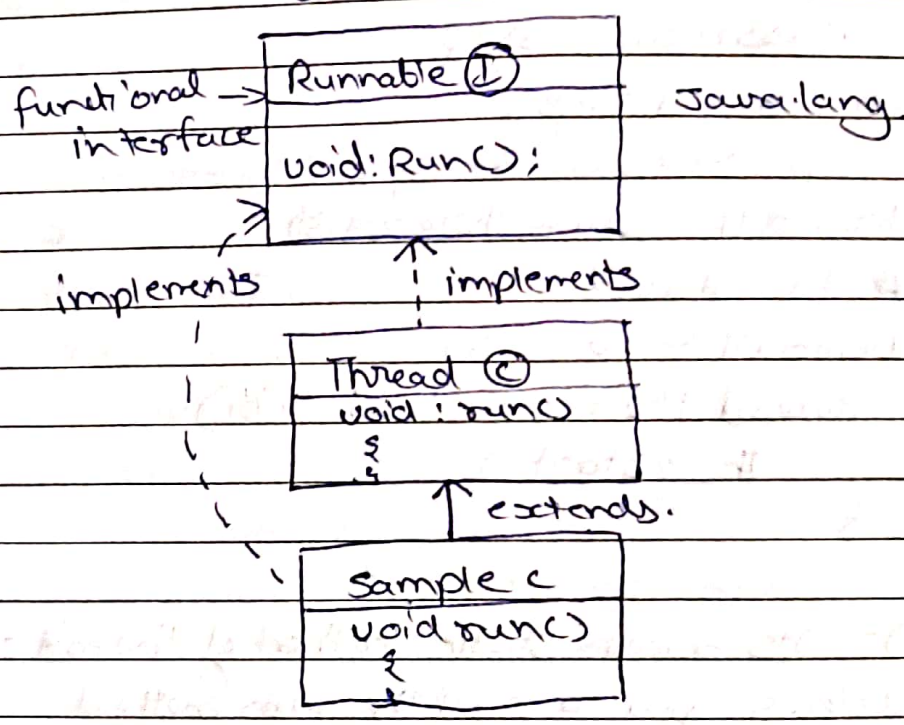
Run method()-

- Run method can be overridden in two ways
1) By extending threads class
2) By implementing Runnable Interface.

- Runnable interface is a functional Interface which belongs to Java.lang package.

Functional Interface - the interface which has only one abstract method is called as functional interface.

eg - interface Same{
void disp();
}.



Multithreading

Processing multiple threads simultaniously is called as multithreading.

Scanned with OKEN Scanner

papergrid

Date: / /

Q- WAP to demonstrate multithreading -

```
Package thread topic;
Class sample extends threads
{
    Public void run ()
    {
      Sop (hi);
    }
Class Demo implements Runnable
  {
     Public void Run()
      {
        sop("hello");
      }
  }.
Public class mainclass1 {
  Psum (str[] args)
   Sample S1 = new Sample ();
   thread t1 = new thread (S1);
   Th1 . Start()
   Demo D1 = new Demo ();
    thread th2 = New thread (D1);
        th2 . start();
  }.
```

Start()- Its a non static method of thread class that allows you to call the Run method implementation directly without actually calling it.

Scanned with OKEN Scanner

For understanding
Consider example of ola you went to book a cab you have put source & destination now you are waiting for the drivers conformation so there comes thread the drivers details like name & phone no is given to you when he confirms the Ride so till the the execution is paused this is threading & yes after 3 minutes if a ride is not confirmed. then it stops & complete the execution this is also thread.

Not synchronized- Processing multiple thread simultaniously with a single shared medium/resource is called as not synchronized.

Synchronized- Processing the threads one by one to a shared resource is called as synchronized.

```
Class sample extends thread
{
Public void Run(){
for(int i=0; i<=10; i++){
    sop(i);
    try{
      Thread.sleep(1000);
    }catch(Interrupted execution.e)
        e.print stacktrace();
      }
    }
} class Demo implements Runnable {
  {
    Public void Run(){
    for(int i=0; i<=10; i++){
```

Scanned with OKEN Scanner

```
sop(i);
try{
    thread.sleep(2000)
  } catch(InterruptedException e){
      Printstacktrace();
  }
class mainclass{
  Psum(str [] args){
    Sample S1 = new sample();
    Thread th = new thread(S1);
    DemoD1 = new Demo();
    Thread th1 = new thread(D1);
      Th.start();
      Th1.start();
  }
}
```

Interrupted exception - It occurs when if as execution goes on sleep for the time then any other program on execution of some code might not access the resource within that time.

```
class shared resource{
  synchronized void res1(){
    for (int i = 0; i <= 10; i++){
      sop(i);
    } try{
        thread.sleep(1000);
      } catch (InterruptedException E){
        e.printstacktrace();
      }
    }
} synchronized void res2(){
```

papergrid

Date: / /

```
for(int i=0; i<=10; i++){
  sop(i);
} try{
    thread.sleep(2000);
  } catch(InterruptedException e){
      e.printstacktrace();
  }
}
} class sample extends Thread{
    SharedResource SR1;
    sample(SharedResource SR1){
        this.SR1 = SR1;
    }
    Public void Run(){
        SR1.Res1();
    }
} class Demo extends Thread{
    shared resource SR2;
    Demo(SharedResource SR2){
    this.SR2 = SR2;
      } Public void Run(){
            S2R.RS2();
      }
Public class mainclass{
  PSVM(str[] args){
  ~~Sample S1= new sample()~~
    SharedResource SR = New Shared Resource();
    Sample S1 = new sample(SR);
    ~~Sample~~ Demo S2 = new ~~sam~~ Demo(SR);
    thread Th = new Thread(S1)
    thread Th2 = new Thread(S2);
      Th.Start();
      Th2.Start();
  }
```

for understanding

- if we do not implement to thread class or runnable interface or we call Run method directly it will execute as as a context to current thread not as a seperate thread.

- if you call a Run method directly like regular method call the code inside the same thread will Run no new thread is created.
if you call using start method a new thread is created & run method is executed in that thread allowing for parallel execution

if Threads class or Runnable interface are not implement then code is not multi threading the methods ~~are~~ marked as synchronized which means that if you use threads concept ~~then~~. (ie thread, Runnable start) then it will run concerently by aquiring the intrinsic lock & releasing it you simply create synchronized method & call them in main method it will get executed sequencly which is not multithreading.

~~dont do like this~~

```
Class hello {
Synchronized public hello(){
    for(int i = 1; i<=10; i++){
        Sop(i);
        try {thread.sleep(1000);
    catch(interrupted exception e){
        e.print stack trace();
    }
Synchronized public hi(){
    for(int i=0; i<=10; i++)
        Sop(i);
    try{
    thread sleep(2000);
catch{Interrupted Exce E){
    e.print stack trace();
}
class mainclass{
    psvm(str[] args)
    {
        hello h1 = new hello();
        h1.hello();
        h1.hi();
    }
```

dont do like this Its not multi threading -

```
class sample extends threads {
synchronized public void run(){
  for(int i=0 ; i<=10; i++){
    sop (i);
    try{
      threads.sleep(1000);
    }catch (InterruptedException E){
      e.printStackTrace();
    }
  }class demo implements Runnable{
    synchronized public void Run()
    for(int i=0 ; i<=10; i++){
      sop(i);
      try{
        threads.sleep(1000);
      }catch(interruptedexception e){
        e.printstacktrace();
      }
    }
```

```
class mainclass -
psum (str[]args){
sample ys4 = new sample4()
thread T1 = new thread()
demo D1 = new Demo()
thread T1 = new thread (D1);
T1.run();
T2.run(); //
```

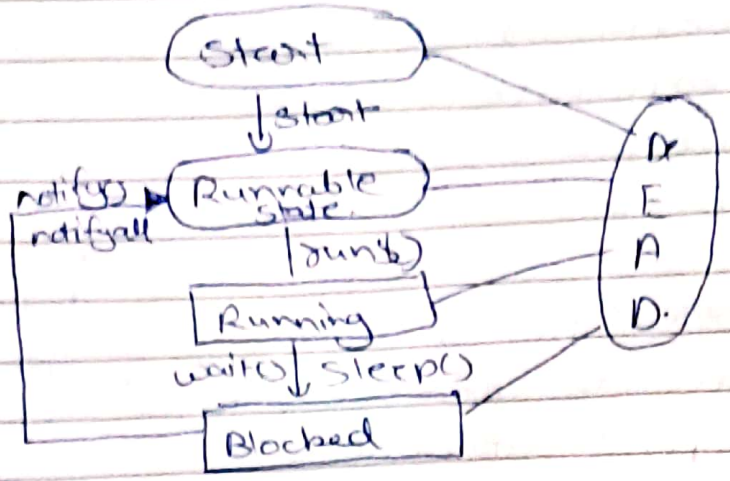
(We dont call Run directly as it will start running as a current thread. not as a seperate thread).

Thread life cycle

It is a life cycle of thread where how it starts its execution & terminate its execution.
initially thread will be in start stage. where it reaches Runnable state. when resources are available it starts executing i.e Running state.
if the program execution gets completed it reaches dead state.

if there is need of resource it will get blocked. & resumes back to Runnable state if the thread is notify.
on any of these stages the thread can reach dead state due to insufficient of resource or wrong programming.

Scanned with OKEN Scanner



understanding

Initially the thread in in start stage then it reaches Runnable state (its like a child waiting for its turn to play) here it waits for the availability of Resource (but here is a chance of deadlock i.e more than one thread might ~~be~~ be pen waiting). now it Resource is available it starts its execution & reaches dead state & if there is need of resource then it waits in blocked state. ~~to available~~ as notify() & when the resource is again available it resumes execution from runnable state.

Deadlock

Deadlock is a situation in computing where two or more threads are unable to proceed because each is waiting for ~~a~~ other to release the resource.

It can reach to standstill in program execution.

notify() - It is a non static method of object class where it just notifies a thread which is about to access the resources.

understanding

it's like tapping a sleeping friend to let him know

that they can use the shared resource it wakes up one waiting thread allowing it to continue task execution when resource is available.

notifyall() - notifyall is a nonstatic method of object class which will notify all the threads which is about to access the resource.

- Understanding - It is like waking up all your friends who were waiting for a shared item it signals all waiting threads that resource is available allowing them to continue.

	wait()	sleep()
1)	Its a non static method of object class.	1) It's a static method of thread class
2)	It is used only in case of synchronized	2) It can be used in any code
3)	It will make the thread wait till it recieves the notification	3) It will pause the execution for few milliseconds

thread class has 3 major properties-

1) **Thread id** - the id is generated by the system.
2) **thread name** - It is the name given for the thread. where we can set & get the name for the thread.
3) **Thread priority** - It is a priority given to the one is the least 10 is the maximum & 5 is default.

current thread() - It is static method of thread class which return the instance of currently executing thread.

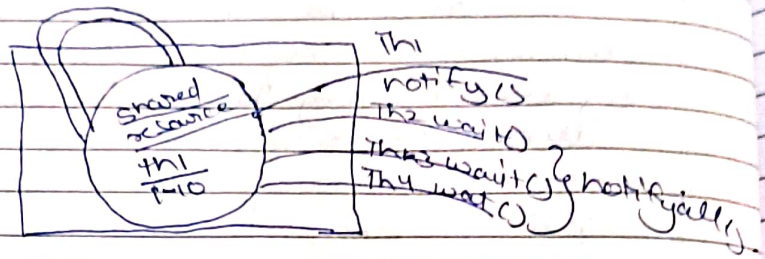
Scanned with OKEN Scanner

for understanding

- On book my show if you book movie ticket you selected seats 20 & 21 till the time you are booking the tickets no one else can book these seats at that particular time if you were not able to book or don't book then after 5 min its still available.

notify & notify all below figure that this.

#



Java skills -

- very Good understanding on constructor.
- good knowledge on method overloading.
- good understanding on method overriding.
- very good understanding on static & nonstatic.
- good knowledge on oops principle
- very good understanding on collections.
- good knowledge on polymorphism

Q. Why Java is object oriented?

Ans- Since Java support class & object, inheritance, encapsulation, polymorphism & abstraction. hence these concepts are supported by Java hence Java is object oriented.

Q. What are the four pillars of Java?

Ans- Encapsulation, Inheritance, polymorphism & abstraction.

INTERVIEW

- on Java we cannot
  only the class which
  as public.
- we can develop a
  inside is called as
  called as outer
  all the 4 types
  be static.

- class cannot be
- we cannot have a
- we cannot inherit
- we cannot inherit
  i.e ~~method~~ constructor name
- we cannot write
  it will give error
  can write, if
  the method is
- you cannot devel
- we can make m
  there is only one
  it at a time
- we cannot use
  main method
  & for accessing
  Superkeyword doesent have
- we can make m
- ~~we doesent prea~~
  ~~abstract it doesent have~~
- we don't c rea
  it has abstract
  a body. It ha

INTERVIEW QUESTIONS.

papergrid
Date: / /

- In Java we cannot create all classes as a public only the class which has the main method can be made as public.
- we can develop a class inside a class the class inside is called as inner class the class outside is called as outer class in inner class we can have all the 4 types of access specifier it can also be static.

- Class cannot be static
- we cannot have a method inside a method.
- we cannot inherit private method.
- we cannot inherit constructor as the first ~~name~~ full i.e ~~method~~ constructor name should be same as class name.
- we cannot write default keyword for method. it will give error in case of switch & interface you can write. if you dont ~~give~~ write access specifier the method is by default considerd as a default method.
- you cannot develop a constructor inside a ~~class~~.
- we can make main method as synchronized as there is only one JVM in JDK. which can access it at a time.
- we cannot use this & super keyword inside the main method as this & super keyword is non static. & for accessing it we need object & this keyword & super keyword dosent have a constructor.
- we can make main method as final.
- ~~we dosent create object for abstract because of the abstract it dosent have a body~~
- we don't create object for abstract because it has abstract method in it which dosent have a body. It has concrete method too.

Scanned with OKEN Scanner

but if it allows to create object then you might can call the abstract method with it so no need of creating object. So because of this there is a Rule The you cannot create an object for abstract.

- ~~We cannot take same variable name in method overloading with different data types. because it will be duplicated compiler always reads var name not data type.~~
- ~~Java dosent allows duplicates~~

- Java dosent allows duplicates

Constructor overloading

developing multiple constructor with the same name but variation in the argument list is called as constructor overloading

Variation in the argument list means-

1) variation in the length of the arguments.
2) variation in the order of occurence of the arguments
3) variation in the datatype.

```
Class Demo{
Demo (int a)
{
 Sop (a);
}
Demo (String a){
    Sop (a);
}
```

```
Demo (int a, String b)
{
  sop (a+" "+b);
}
Demo (str a, int b)
{ sop(a+" "+
}
} class sample
{
  psvm (Str[]
  {
  Demo D1= ne
   Demo D2= n
   Demo D3=
   Demo D4= n
  }
}
```

Const

one constructor c
constructor chain

- to achieve const
1) this() :- (this c
2) Super(); - (super

- this() - when eve
within the cla
statement.

Rules- 1) it sho
const
2) we
st

```
Demo(int a, string b)
{
  sop(a+" "+b);
}
Demo(str a, int b)
{ sop(a+" "+b);
}
} class sample
{
  psvm(str [] args)
  {
  Demo D1 = new Demo(10);
   Demo D2 = new Demo("Hi");
   Demo D3 = new Demo("10", "Hiii");
   Demo D4 = new Demo("Hi", 10);
  }
}
```

we have to create multiple object this will aquire more memory to overcome this we go for constructor chaining.

Constructor chaining

one constructor calling another constructor is called constructor chaining.

- to achieve constructor chaining we have two statements-

1) this();- (this calling statement)
2) Super();- (super calling statement)

- this()- when ever we achieve constructor chaining within the class we should use this calling statement.

Rules-1) it should be always first statement inside a constructor
2) we cannot develop more than 1 this calling statement inside the constructor

Scanned with OKEN Scanner

```
Class Demo{
  Demo(int a){
    Sop (a);
  }
  Demo (String b){
    this(10);
    Sop(a);
  }
  Demo (int a, String b){
    this("hii");
    Sop (a+" "+b);
  }
  Demo(String a; int b){
    this (10, "hii");
    Sop(a+b);
  }
} Class sample{
  PSVM (STR[] args){
  Demo d1 = new Demo ("hello", 10);
  }
}
```

O/P= 10
hii
10, hii
hello, 10

Super calling statement (super();):-

Whenever we achieve constructor chaining between the classes we have to use ~~constructor chaining~~ super calling statement.

Rules-

- It should be the first statement inside the constructor
- we cannot develop more than 1 super calling statement.
- Mandatorily there should be is a relationship.

Scanned with OKEN Scanner

Note- 1) we cannot develop both this calling & super calling in a same constructor.
2) each & every constructor will by default have super calling statement if we develop this calling statement compiler will not add supercalling statement.

```
class tester{
    tester(int a){
        sop(a);
    }
}class driver extends tester{
    driver(String a){
        super(10);
        sop(a);
    }
}class mainclass{
    PSVM(Str[]args){
        Driver D1 = new Driver("hii");
    }
}
```

o/p = 10
Hii

Diamond problem / Ambiguity issue

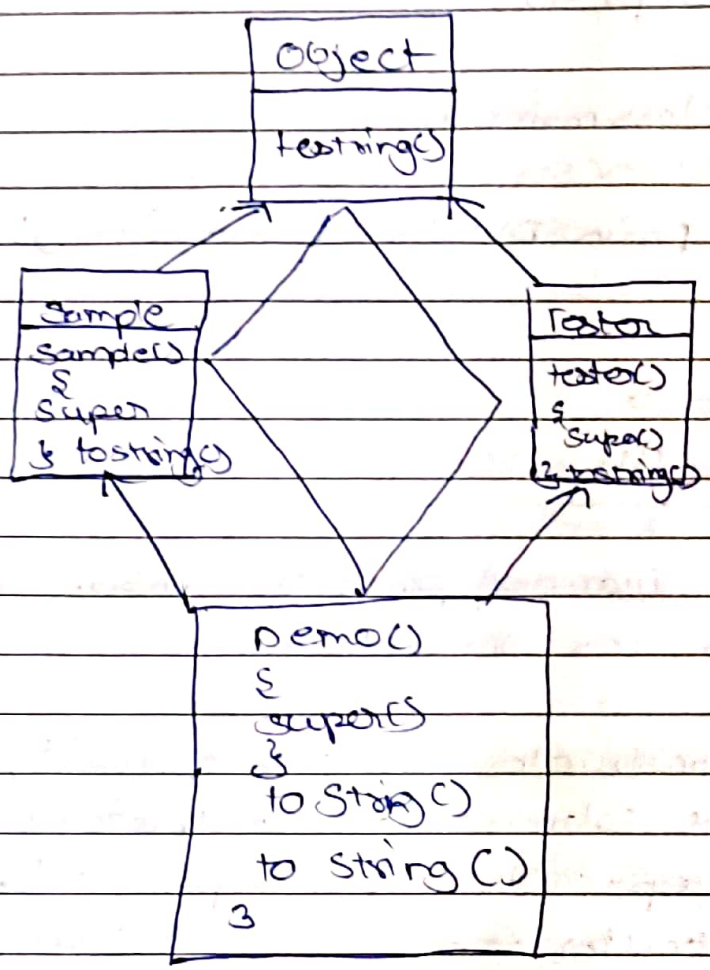
the main reason for diamond problem is:

because of constructor- in case of multiple inheritance the submost class constructor will get confused exp- if there is a super calling stateme in the class constructor from which we have passed an argument so which constructor has to be invoked will cause ambiguity.

Scanned with OKEN Scanner

- **Because of object class** - in case of multiple inheritance submost class will get confused to inherit the properties of object class.
- as both superclasses will inherit properties from ~~sup~~ object class consider ~~[illegible]~~ situation in which you have overridden the method of object class in both superclass & you inherit those in the submost class & you create object for submost class & try to call that method with it now the compiler will get confused which method to call.

So because of this we have interface.



* -> interface.
x -> constructor
x -> object

papergrid
Date: / /

Wrapper class

wrapper classes are the inbuilt classes for each & every primitive data type.

all the wrapper class belongs to java.lang package.

boxing- converting primitive datatype to its corresponding wrapper class object is called as boxing.

unboxing- converting from wrapper class object to corresponding primitive data type is unboxing.

Note- boxing & unboxing is done by the system implicity.

wrapper class

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

```
int a = 10
Integer I = new Integer(a); //Boxing
int b = i; //unboxing

boolean 2 = true;
Boolean 2 = new Boolean(2); //boxing
int b = i; // unboxing.
```

Scanned with OKEN Scanner

Java

Java is an object oriented programming language but it has primitive datatype due to which it is not 100% object oriented as primitive datatypes are keyword they do not have constructors so we cannot create object for it. So to overcome this we have wrapper class for each primitive data type this makes Java 100% object oriented.

Singleton design pattern

avoiding multiple time object creation throughout the life cycle of an application is called as Singleton design pattern.

- to achieve singleton design pattern declare the constructor as private & go with indirect access like getters & setters.

```
Class Demo{
  int a=10;
  Static int count =0;
  Static Demo D;
  private Demo(){
      count ++
  }
  Public static Demo getobject(){
      if(count <=0){
          D= new object()
      } Return D;
  }
```

Scanned with OKEN Scanner

```
public class mainclass
{
psvm(str[] args){
    Demo D1= Demo.getobject();
        Sop(D1);
    Demo D2= Demo.getobject();
        Sop(D2);
    Demo D3 = Demo.getobject();
        Sop(D3);
    }
}
```

Length - It is a predefined variable used to find array's length.
- It is present in Arrays class
- It belongs to Java.util package.

alternate name
variable/data member/attribute

Comparable

1) Comparable is an interface that belongs to Java.lang package
2) Comparable is a functional interface i.e it has only one abstract method
3) whenever the class is not final we should go with comparable
4) whenever we need to sort the object based on object value we go for comparable.

Syntax-

```
Public int compareto(object obj);
```

- we can have only 1 return statement in a method
  & Ambiguity/confusion.

papergrid
Date: / /

Note:- we can store other datatypes in an array by creating the array to classtype.

```
Public class main{
psvm (str[] args){
 object arr[] = {10.5, true, 78, "shahbansir", 'a'}
}
```

& datatypes have wrapper class so it internally boxed to its wrapper class object

```
Class Hero implement comparable{
 String name;
 int age;
 double sal;
 Hero(String name, int age, double sal){
  this.name = name;
  this.age = age;
  this.sal = sal;
 }
 Public int compareto (object obj){
   Hero HS = (Hero) obj
   return this.age - HS.age; >> sorting in ascending order
 // return HS.age - this.age; >> Sorting in decending order
 // return(int)(this.sal - HS.sal); >> based on income.
 // return this.name compareto (HS.name); >> sorting
                 based on name in ascending order.
 return HS.name compareto (this.name) >> sorting
      based on name in decending order.
 }.
 Public string to string( ){
  return this.name + " " + this.age + " " + this.sal;
 }
```

```
public class mainclass {
psum(Str[] args) {
Hero H1 = new Hero("dboss", 999.9, 49);
Hero H2 = new Hero("Rajkumar", 777.7, 99);
Hero H3 = new Hero("yash", 555.5, 99);
HeroArr[] = {H1, H2, H3};
Arrays.sort(Arr); //-> sort() will internally
call compareto(object obj)
// fetching all the values from an array one
by one using for loop
for(int i = 0; i < arr.length; i++) {
    Sop(arr[i])
}
}
}
```

compareto()- it is a method of string class. which is used to compare the string based on characters.

- if we have a parameterised constructor & we don't pass arguments while creating object init, then we will get compile time error.
- we do downcasting while overriding compareto method as we have upcasted and the control is in superclass so we downcast as the implementation is in subclass so we need to downcast.

understanding

Ctrl + Shift + F

in this value are fetched like the current object becomes arr[i] & comparing object its becomes [arr++i] and based on the o/p arrays.sort method internally sorts the array using quick sort algorithm.

as current next element
arr[i] [i+1]
element

papergrid
Date / /

Comparator

- It is a predefined interface which belongs to Java.util package.
- when the class is final then we have to go with comparator
- wherever we need to ~~compare~~ sort the object based on object values we go with comparator.
- comparator is not a functional interface.

Syntax: (public int compare (object o1, object o2))

```
final class hero{
    string name;
    int age;
    double sal;
    hero (string name, int age, double sal){
        this.name = name;
        this.age = age;
        this.sal = sal;
    }
    Public String toString(){
        return name + age + sal;
    }
}
class Demo implements comparator{
    Public int compare (object o1, object o2){
        hero h4 = (hero) o1;
        hero h5 = (hero) o2;
    return h4.age - H5.age; > sortin in ascending ord
// return H5.age - H4.age; > descending order
// return (int) (H4.sal - H5.sal); > based on sal
// return H4.name compare to Ignore case (); name
```

```
}
public class mainclass {
psvm (String [] args) {
heroH1 = new hero("sharbarsir", 18, 99.9);
heroH2 = new hero("shashankbsir", 70, 89.9);
heroH3 = new hero("Rahulsir", 60, 100.5);
hero arr[] = {H1, H2, H3};
Arrays.sort(arr, new Demo());
for (int i = 0; i < arr.length; i++) {
        Sop(arr[i]);
}
}.
```

for understanding

if happens like object1 take i element & object2 is having ~~object of Demo class so it can access i+1 element array~~ object of Demo class so it can access the array of object internaly so fetching happens like h1 has object at arr[i] & h2 has arr[i+1] simultaniously & than values are subtracted if result is negative then the values are swapped. the o/p is 1 for positive -1 for negative & 0 for zero.

Class loder types

1) System class loader - It will load all the user defined classes and interface.
2) Bootstap class Loader - It will load all the predefined classes and interface.
3) extension classloader - It will load all the external ~~class~~ classes or external Jag files.

Scanned with OKEN Scanner

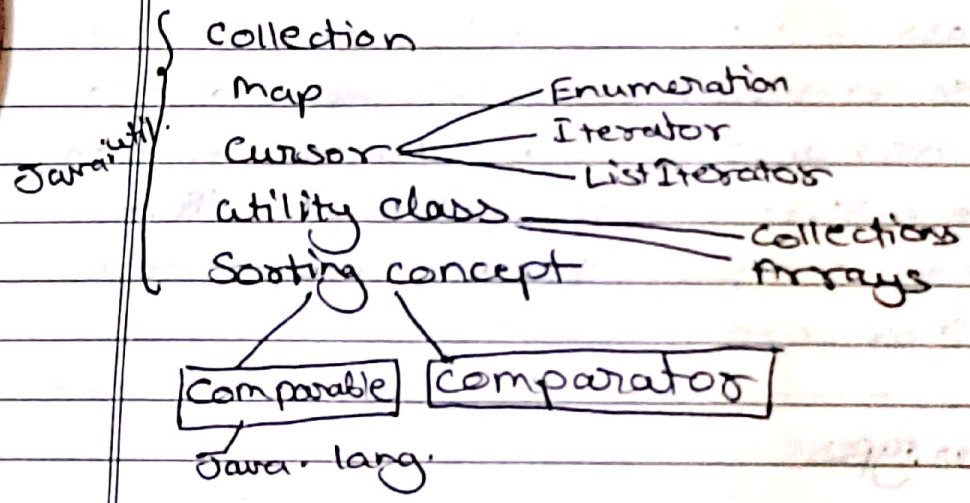
Collection Framework

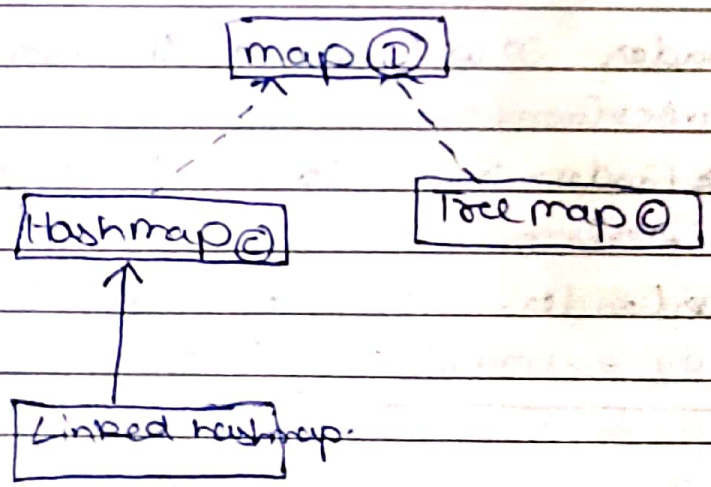
collection framework is a unified architecture which contains both classes and interface.

Collection-

- It is a predefined interface which belongs to Java.util package.
- Whenever we want to store both homogeneous & heterogeneous type of data the nue should go with collections.
- In collections size is dynamic.

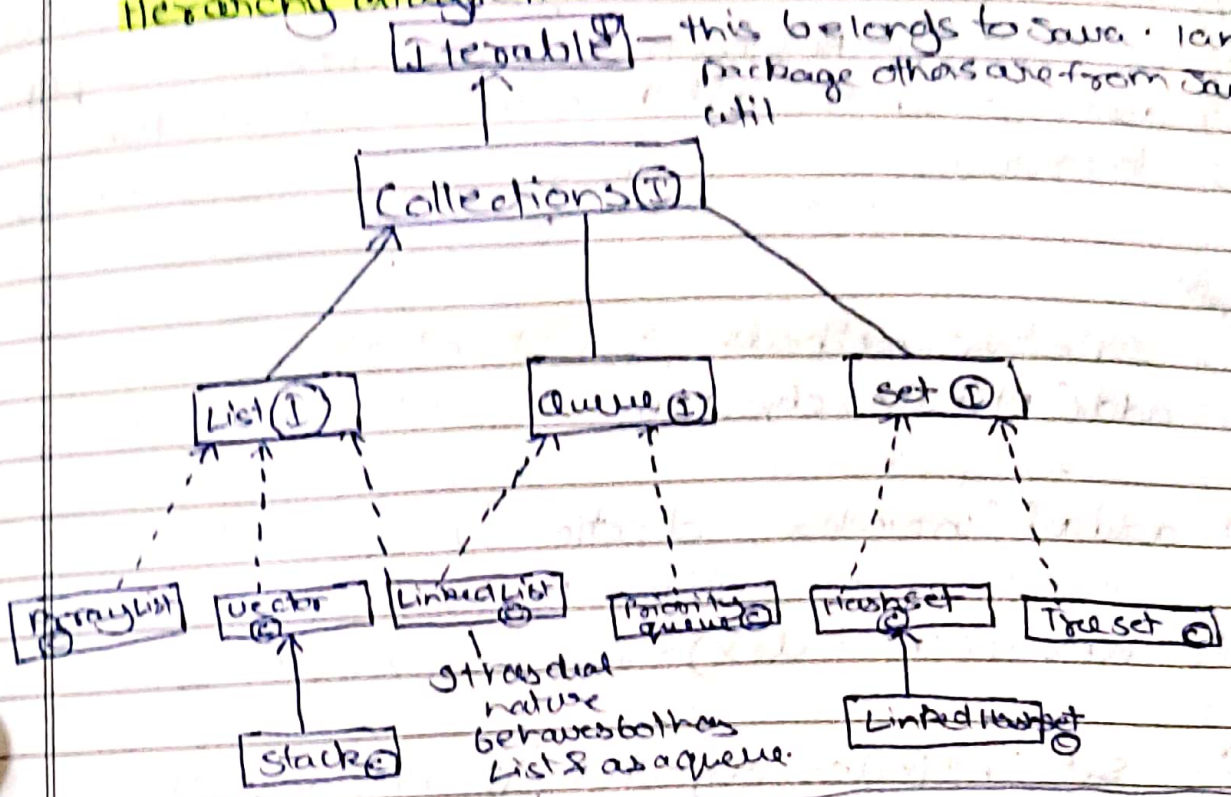
Collection framework





Scanned with OKEN Scanner

Hierarchy diagram-



important methods of collections interface-

(have abstract methods)

1. add (object obj) - used to add object to a collection.
2. addAll (collection C) - used to add the elements from a specified collection
3. remove (object obj) - used to remove object element from collection.
4. removeAll (collection C) - Removes all the duplicate elements specified from the collection that are also in the collection
5. contains (object obj) - Returns true if element is present.
6. ~~collection~~ containsAll (collection C) - Returns true if the collection contains all the elements from specified collection
7. retainAll (collection C) - It is used to remove the unique value from collection & keep only the value which are present in specified collection. if modification is done then it gives true.

Scanned with OKEN Scanner

8) size() - Returns the no of elements in the collection.
9) is empty() - Returns true if the collection is empty.
10) clear() - remove all the elements from collection
11) to Array() - Returns an array of all the elements in the collection.

All are abstract methods

important methods of List interface -

① add(int index, object obj) - used to add a element to specific index.
② addall(int index, collection c) - add all the elements of collection to specified index value.
③ remove(int index) - remove element from specified index
④ set(int index, object obj) = Replaces the element at that index with the new element.
⑤ get(int index) = Return the index at the specific position.

Q1 -
```
import Java.util.ArrayList;
Public class mainclass{
 Psvm (str [] args){
 ArrayList A1 = New Arraylist[];
        A1.add(25);
        A1.add(25.5);
        A1.add("hello");
        A1.add (true);
        A1.add (a);
      Sop(a)
  }
}
```

O/P [25, 25.5, hello, true, a];

papergrid

Date: / /

```
import Java.util.ArrayList.
Public class sample{
psvm (str[]args){
ArrayList A1 = new arraylist();
  A1.add(25);
  A1.add("hello")
  A1.add("true");
  A1.add("a");
ArrayList A2 = new ArrayList();
  A2.add(45);
  A2.add(125);
  A2.add("bye");
sop(A1);
  sop(A2);
  a1.AddAll(1,a2);
    sop(a1);
    sop(a2);
```

O/P [25, hello, true, a]
[45, ~~hello~~ 125, bye].
[25, 45, hello, true a, 45, 125 bye]
[45, 125, bye]

```
import Java.util.ArrayList
class sample{
ArrayList A1 = new ArrayList();
A1.add(25);
A1.add("hello),
A1.add(true);
A1.add('a');
ArrayList A2 = new ArrayList();
  A2.add(25);
  A2.add(45);
  a2.add("bye");
a1.removeAll(a2);
sop(a1);
}
}
```

O/P [hello, true, a];

```
import Java.util.arrayList;
Class sample{
psvm(str[]args){
ArrayList A= new arrayList()
a.add(25);
a.add("hello);
a.add(true);
sop(a);
a.remove(0);
sop(a)
```

O/P
[25, Hello, true]
[Hello, true]

```
import Java.util.ArrayList;
class Samples
psum (str [] args){
ArrayList a1 = new ArrayList();
  a1.add(25)
  a1.add ('hello')
  a1.add (true);
ArrayList a2 = new arrayList();
  a2.add (25)
  a2.add("hello)
  a2.add(true);
  a2. add('bye');
  a2. add (25);
Sop(a2.containsAll(a1))
}
}
```

o/p = true.

```
import Java.util.ArrayList;
class sample{
psum (str [] args){
ArrayList a1 = new arrayList()
a1.add(25);
a1.add ('hello');
a1.add (true);
a1.add ('a');
ArrayList a2 = new arrayList();
a2.add (45)
a2.add (25);
a2.add(bye);
a1.retainsAll (a2);
sop (a1);
sop(a2);
}}
```

o/p [25]
[45, 25 bye]

```
import Java.util.ArrayList
public class sample{
psum(str[] args){
ArrayList a = new arrayList();
  a.add(25);
  a.add("hello");
  a.add (true);
  a.add('a');
  a.add (null);
  sop(a)
  a.set(1, 2.6).
  sop(a)
}}
```

o/p [25, hello, true, a, null]
[25, ~~hello~~, true, a, null]

```
class sample{
psum(str[] args){
ArrayList a1 = new arraylist(
  a.add(25);
  a.add("hello");
  a.add(true);
  a.add(a);
  sop(a);
for(int i=0; i<a.size(); i++){
  sop(a.get(i));
  }
}.
```

o/p- [25, hello, true, a]
25
hello
true

```
import Java.util.arrayList
import.Java.util.arrays
public class sample{
psum(str[]args){
ArrayList a2 = new ArrayList();
  a2.add(25); //object o= new Integer(6)
  a2.add("hello");//object o= "hello
  a2.add(true);//object o= new Boolean(true)
  a2.add('a'); object o= new Character(a)
  Sop(a2)
   object[]arr = a2.to array();
    Sop (Arrays.toString(arr));
}
}
```

o/p [25, hello, true, a]
[25 hello, true, a]

```
import Java.util.ArrayList
class sample{
 psvm (str[] ao){
ArrayList a= new ArrayList();
 a.add(25)
 a.add(2.5);
 a.add ('hello');
 a.add(true);
 Sop(a);
 a.remove("hello");
 Sop(a);
Sop(a.size());
 Sop(a.isempty);
 a.clear();
 Sop (a);
 Sop(a.isempty))
}
}
```

o/p [25, 2.5, hello, true]
[25, 2.5, true]
3
false
[]
true.

Array List

- Array List is a predefined class which belongs to Java.util packag
- its a subclass of list interface.
- In array List we can store both homogeneous & heterogeneous values
- In Array List size is dynamic.
- the size will get increased by 50%.
- formula $\left(\left(c \times \frac{3}{2}\right) + 1\right)$
- default capacity will be 10.
- In array list we can add null values
- In array List we can add duplicate values.
- Array List follows order of insertion.
- All the methods in array List are non synchronized. So the performance will be high but there will be

Scanned with OKEN Scanner

cc-current capacity.

papergrid
Date: / /

data mismatch.

- Each and every ~~word~~ value will be stored in linear data structure format.
- In case of fetching the obj of arraylist is the best choice.

Vector

- vector is a predifined ~~interface of~~ class which belongs to Java.util package.
- vector is a subclass of list interface
- In vector size is dynamic
- In vector size is increased by 100% (cc*2)
- ~~In~~ vector is also called as legacy class
- Default capacity of vector is 10.
- we can store both homogeneous & heterogeneous values in vector
- we can add ~~both~~ null values
- we can add ~~default~~ duplicate values.
- It follows order of insertion.
- each and every value is stored in linear data structure format.
- All the methods are synchronized so performance is high but there is no data mismatch.
- In case of fetching of object vector can be a ~~size~~ choice but we dont use it as ~~it will~~ when we will add extra object size will be increased by 100%.

Scanned with OKEN Scanner

```
import java.util.vector
class sample {
  psvm (str[] args){
vector v = new vector();
  v.add(10);
  v.add(10);
  v.add("hello");
  v.add("null");
  sop(v);
  sop(v.capacity());
}
}
```

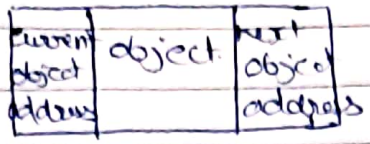
o/p - [10, 10, hello, null]
10

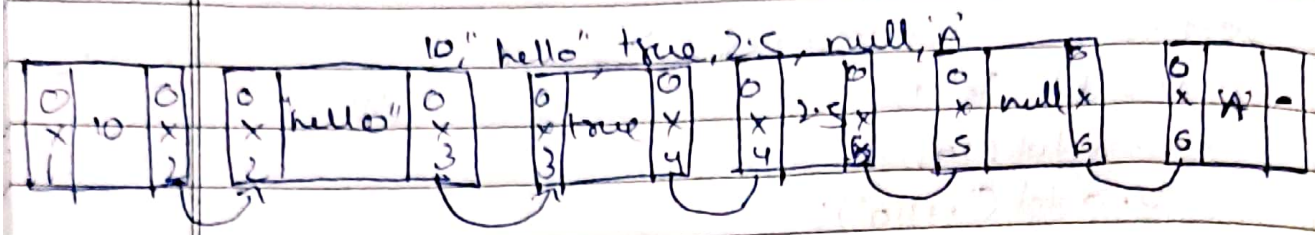
// capacity() - it is a method of vector class which gives capacity of collection.

Linked List

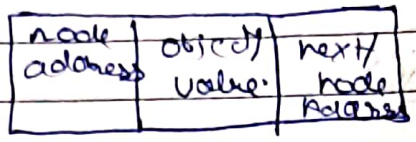
Its a predefined class which belongs to java.util package
Its a subclass of both List & Queue interface
In Linked List size is dynamic
In Linked List default capacity is 10
In linked list size will get increased by 50%.
formula = $\left(CC \times \frac{3}{2}\right) + 1$
we can add null values.
we can add duplicate values.
we can add both homogeneous & heterogeneous type of data in Linked List.
It follows order of insertion
each & every value will be stored in node format
In linked list all the methods are ~~in~~ ~~node~~ not synchronized i.e performance is high but data mismatch is there
In case of insertion and deletion Linked List is the best choice.

Scanned with OKEN Scanner





syntax-



for understanding

linked list is like line of people holding hands where each person knows who's next, it is easy to find someone, add new people or let someone go because of the handholding connection.

the first node of linked list is called head.

Adding and removing is easy because you don't have to ~~traverse~~. move a lot of stuff like in array. like if you remove the first element from the array all the index will be shifted in linked list in linked list adding a element is like holding the hands of person in front you don't have to shift others & if you have to add at last. traverse the the last node and add a new node and make the current node to point at new node.

- & for removing the element in linked list to remove the first element just update the head to the second node in the list & element is removed & for removing from last traverse till the second last node then update the reference of the next object address to null.

Peek()

```
import Java.util.linkedlist
class sample10{
 Psvm(str[] args){
Linked List L1= new LinkedList()
l.add(10);
l.add("hello");
l.add(true);
l.add(2.5);
l.add(null);
l.add(A);
Sop(l.peek())
Sop(l);
}
}
```

o/p - 10
[10, hello, true, 2.5, null, A]

Poll()

```
import Java.util.linkedlist.
class sample2{
Psvm(str[]args){
LinkedList L = new LinkedList();
L.add(10);
L.add("hello");
L.add(true);
L.add(2.5);
L.add(null);
L.add('A');
Sop(L.poll());
Sop(L);
}
}
```

o/p - 10.
[hello, true, 2.5, null, A]

Note - Peak & poll are the important method developed in queue interface.

→ Peak is used to fetch the first element but it won't remove it.

→ Poll is used to to fetch the first element but it will Remove it.

Priority Queue

- Priority Queue is a predefined class ~~of~~ belongs to Java.util package.
- Priority Queue is a subclass of ~~list~~ queue interface.
- In priority queue size is dynamic
- In Priority Queue size is increased by 50% $\left(\left(\frac{c \times 3}{2}\right)+1\right)$
- the default capacity of priority queue is 11
- ~~Size will get increased~~ In priority queue we can add homogeneous value
- In priority queue we can add dupliate value. In priority queue we cannot add null value.

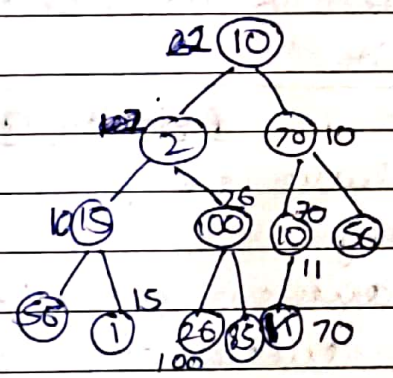
Scanned with OKEN Scanner

- Priority queue is partially autosorted or doesn't follows order of insertion.
⇒ each and every value is stored in tree structure
⇒ All the methods in priority queue is not synchronized ie performance is high but there is data mismatch

understanding Priority queue is partially autosorted so that the element with highest priority (top) is always at front & easily accessable. (priority depends upon the implementation (how data is entered) & what is the specific requirement of the application) (like how you want to retrive the data i.e from forward or backward or any way etc)

- class Sample
{
Psvm(str[] args)
{
Priority queue P = new priority queue();
P.add(10)
P.add(2)
P.add(70)
P.add(15)
p.add(100);
P.add(10);
P.add(56);
P.add(56);
P.add(1);
P.add(26);
P.add(85);
P.add(11);
Sop(P);
for (object a : p){
sop(a);
}
}
}

[1, 2, 10, 10, 26, 11, 56, 56, 15, 100, 85, 70]



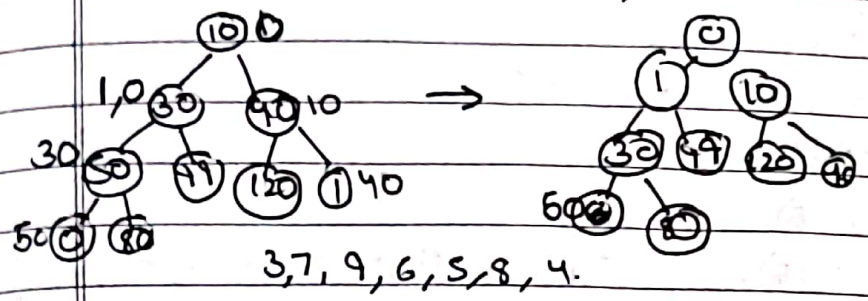
1
2
10
10
26
11
56
56
15
100
85
70

compare the value of head & branch if branch is less than head the swap the value.

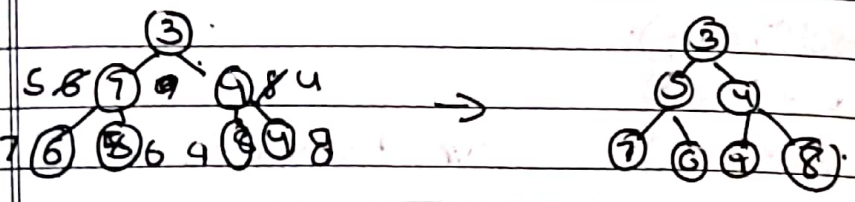
Scanned with OKEN Scanner

for understanding - here we have used for each loop to iterate over the element of the collection
we did Object A because the type of element in priority queue is Object as we have not specified the type of parameter & we are using raw priority queue.
It is going into the collection storing the elements of the collection in variable & & printing it

10, 30, 40, 50, 44, 120, 1, 0, 80



3, 7, 9, 6, 5, 8, 4.



Hash set:

- It is a predefined class which belongs to Java.util package
- It is a subclass of set interface
- In hashset size is dynamic
- the size is increased by 50% $\left(cc \times \frac{3}{2}\right)+1$
- The default capacity of hash set is 16
- We can add both homogeneous & heterogeneous values.
- We can add null values.
- Hashset doesn't allows duplicate.
- It doesn't follows order of insertion.
- All the methods are not synchronized i.e performance is high but there is data mismatch.

for understanding

hashset do not sort as its an unordered collection of Java. (it can add heterogeneous values so sorting is not possible) & yes the order of insertion is not followed it it. as it stores data in hash table format.

- When you add an element then it calculate the hashcode for it using hascode() method & add the data into the

Why duplicates are not allowed in hashset - so there is a linked list being maintained of the element that are to be stored in the hashset. So what happens is a hashcode is generate for an object & if the hashcode of one object matches with the other then the comparison is done using equals() of object

for understanding

- It allows efficient adding & removal of data.

```
import Java.util.hashset
Class sample{
  Psvm(Str[] args){
  HashSet H1 = new Hashset();
    H1.add(10);
    H1.add(2.25);
    H1.add(2.25);          o/p = [null, 2.25, hello, 15, true]
    H1.add("hello");
    H1.add(null);
    H1.add(true);
    Sop(H1);
  }
}.
```

Linked Hash set

- Linked hashset is a predefined class which belongs to Java.util package.
- Linked hashset is a subclass of Hashset
- Size is dynamic
- Default size of linked hashset is zero
- Size is increased by 50% $(cc \times \frac{3}{2}) + 1$;
- We can store both homogeneous & heterogeneous value in Linked hashset.
- We can add null values.
- It doesnt allows duplicate.
- It follows order of insertion.
- all the methods in Linked hashset are not syncronized ie performance is high but there is data mismatch.

for understanding

When we need a collection that guarantee both uniqueness & maintains the order in which elements were added then we go for Linked hashset.

what happens ... list of the elements in order they were inserted which allows it to iterate over the elements in the order they were inserted

papergrid
Date: / /

```
import Java.util.LinkedList
Public class sample{
Psum(Str[] args){
Linked hashset H1 = new Linked hashset();
    H1.add(15);
    H1.add(2.25);
    H1.add(true);
    H1.add("hello");
    H1.add(null);
    H1.add(2.25);
    Sop(H1);
  }
}
```

o/p [15, 2.25, true, hello, null]

TREE SET

- It is a predefined class which belongs to Java.util package
- It is a subclass of set interface.
- In tree set size is dynamic.
- default capacity of tree set is 16.
- size is increased by 50% formula $\left(cc \times \frac{3}{2}\right)+1$.
- ~~use~~ In tree set we can store homogeneous we can't store heterogeneous values.
- we tree set we can't Store null values.
- we cannot store duplicate values.
- treeset is completly autosorted.
- tree set doesent follows order of insertion.
- All the method in tree set are not synchronized ie performance is high but there can be data mismatch.

for understanding -

tree set stores data in Red black tree format.
↓
balanced binary search tree

```
import java.util.treeset;
class sample
{
  psvm(str[]args)
  { treeset p = new treeset();
        p.add(9);
        p.add(2);
        p.add(80);
        p.add(70);
        p.add(40);
        p.add(210);   o/p = [2, 9, 70, 80, 210].
        sop(p);
    }
}
```

for-each loop

- It is also called as enhanced for loop Specially designed ~~for~~ to iterate over arrays or collection
- It is very simple to use it.
- In for each loop we cannot perform backward direction of fetching.

Syntax - for(datatype variablename : array/collection variable)
{

}

```
int[] number = {1, 2, 3 4 5}
  for(int num : number){
        sop(num);
  }
```

it will iterate the array sequencially & assign the value to variable. & prints it. it automatically iterate over the collection or array doesn't requires to enter

Generics

- Making the class as homogeneous type is called as generics.
- to achieve generics we need angular bracket.
- to achieve generics we can use only classes or non primitive datatype.

```
import Java.util.ArrayList;
~~Public class sample{~~
~~psum(str[]args){~~
~~ArrayList A1 = new ArrayList();~~
~~A1.add~~ Public class sample{
        Public sum(str[]args){
        ArrayList<Integer> a = new ArrayList<Integer>();
            a.add(25);
            a.add(2);
            a.add(30);
            a.add(78);
            Sop(a);
        }
    }
```

it is not necessary

Note- whenever we take wrapper class we take the lowest Range (i.e byte) of primitive datatype.

Q1

```
Pclass sample
PSUM(str[] args){
  Integer a = 100;
  Integer b = 100;
  Integer c = 128;
  Integer d = 128;
  if (a==b){
   sop("hello"){ else{
    sop("bye");
   }
  if (c==d){
   sop("hi");
  }
  else{
   sop("Rod");
  }
}
// for integer wrapper class ranges between -128 to 127 lowest range

O/p = hello, Rod.
```

in wrapper class to optimize memory. If objects value & it lies within the range same then only a single object is created.

```
Public Class sample{
    PSVM(str[] args){
        byte b = (byte)300;
        Sop(b);
    }
}
```

O/p = 44.

the range is from -128 to 127 so it will just cut the whole range to bring it down & within the range and wrap it

other way is
Shortcut: 300 % 256 = 44
~~200 % 256 = 56~~
600 % 256 = 88

Byte ranges from -128 to 127.
So total = 256 char.

Byte 300 = 300
-256
44

in a case Byte B = (Byte) 200
200
-256 O/p = -56.
-56.

Sob ~~ojien~~ no range 256 se ~~nam~~ kaha ve sign.

in case of negative. Byte B = (Byte) -200
-200
-256 same just sign range
-56

There will be internal data loss.

Cursors

Whenever we want to fetch the object from the Collection one by one then we should go with cursors.

Types of cursors

ctrl+shift+O - to remove all unused package.
Alt+shift+R - to replace all value.

papergrid

Date: / /

Enumeration -

- Enumeration is also called as legacy cursor which is only applicable for vector class.
- By using enumeration we cant fetch the object from Backward direction.

for understanding - for enumeration we have to use elements()

enumeration is an interface of java.util package.

- → has more Elements () - returns true if more elements is found & false if not.
- → next element () - it move the cursor. the next element.

for these methods vector class will provide implementation

Enumeration e = v.elements()

interface

enumeration is return type for elements() - (actually the return type is object of the inner class that provided the implementation for enumeration instance)

```
import Java.util.Enumeration;
import Java.util.vector;
public class main.
{ psvm(str[] args){.
    vector<Integer> v = new vector<Integer>();
        v.add(25);
        v.add(30);
        v.add(55);
        v.add(5);
    Enumeration e = v.elements();
        while (e.hasMoreElements()); {
            sop (e.next Element());
        }
    }
}.
```

Iterator - Iterator is also called as universal cursor - which is applicable for all classes of collection interface.

- We cannot fetch the object from the back ward direction

for understanding: Iterator() belong to collection interface so applicable for all the classes.

It has methods like - hasnext() - to check if next element is present.

next() - to move cursor to next element.

as enumeration was only applicable for vector so this was introduced

```
import Java.util.Iterator;
  import Java.util.priorityqueue;
Public class sample
  {
  Psvm(Str[] args){
   PriorityQueue<Integer> = new priorityqueue<Integer>
          a.add(25);
          a.add(30);
          a.add(55);
          a.add(30);
        Iterator i = a.iterator();
        while (i.hasnext()).
          {
            Sop(i.next());
          }
    }
}
```

op:- 25
30
55.
30

List Iterator - By using list iterator we can fetch the object from both the direction.

- List Iterator is only applicable for list interface
- First we should go in forward direction then we should fetch is backward direction.

```
import Java.util.LinkedList
import Java.util.List iterator.
public class sample10
{
  psvm(str[]args){.
  LinkedList<Integer> a = new LinkedList<Integer>();
        a.add(25);
        a.add(30);
        a.add(55);
        a.add(3);
  ListIterator i = a.ListIterator();
   while (i.hasnext()){
        Sop(i.next()).
   }
   sop("------")
   while (i.hasprevious()){
   {
        sop(i.hasprevious());
   }
}
```

o/p - 25
30
55
3

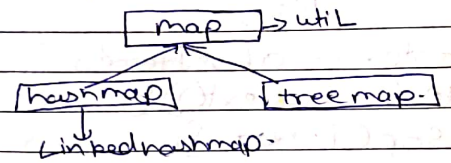
3
55
30
25.

papergrid

Date: / /

Maps

- map is a predefined interface which belongs to java.util package.
- maps are by default generics.
- size is dynamic
- On map everything will be stored in entry format.
- whenever we want to store data upon key-value pair we should go with map.
- In the map we can add duplicate value but not duplicate keys
- ~~map~~ On the map we cannot add null keys but we can add null values.
- map has two subclass 1) hashmap 2) treemap. hash map has one more subclass LinkedHashmap



entry pair format - it means key & value pair format each entry in map is represents the key & value pair.

key - it is unique identifier used to access stored data
value - it is the data associated with key

important methods of map -

1) Put(key, value) - it is used to add a new entry or new key & value pair into the map.

2) PutAll(map) - it is used to copy all the entries from one map to another map.

3) size() - it is used to check how
4) clear() - it is used to remove all
5) is empty() - it gives boolean
no entries into the
6) get(key) - it is used to fetch
with the specific
7) getkey() - used to get the
8) getvalue() - used to get the
9) collections.emptymap() - it
immut
10) contains key(key) - it is used to
key is prese
11) contains value(value) - it ch
in the
12) remove(key) - it is used to
for the spec

Hash map

1) It is a predefined class
package.
- It is a subclass of map
- Hash maps are by default
- hash map size is dy
- On hash map we can
we can add duplica
- we can add both
- It doesn't follows
- all the methods are
for understanding

Map.Entry - It is an int
interface. it provides
and value pairs. &
entryset() - it returns
by entryset() will give
implementation

(3) size() - it is used to check how many ~~elements~~ entries are there in map
(4) clear() - it is used to remove all the entries from the map
(5) is empty() - it gives boolean value true if there are no entries into the map & false if there is any.
(6) get(key) - it is used to fetch the value associated with the specific key.
(7) getkey() - used to get the keys
(8) getvalue() - used to get all the values in the map. } Present in Entry interface.
(9) collections.emptymap() - it will return an empty & immutable map.
(10) contains key(key) - it is used to check if the specified key is present in the set or not.
(11) contains value(value) - it checks if the value is present in the specified map or not.
(12) remove(key) - it is used to remove key & value pair for the specified key.

Hash map

1) It is a predefined class of ~~map interface~~ Java.util. package.
- It is a subclass of map interface.
- Hash Maps are by default generics.
- hash map size is dynamic
- In hash map we cannot add duplicate keys but we can add duplicate values.
- we can add both key & value as null.
- It doesn't follow order of insertion.
- all the methods are not synchronized.

for understanding

Map.Entry - It is an interface in map. interface, or inner interface. it provides methods to access key and value pairs & perform operation on them.

entryset() - it returns a set of map entry - the object returned by entryset() will invoke the inner class which has provided implementation for the entry interface.

in map for long wrapper class we should put l at the end

papergrid

Date / /

```
import Java.util.collections;
import Java.util.~~collections~~ Hashmap;
import Java.util.map;
import Java.util.entry;
public class hashmap{
  psvm(str[] args){
Hashmap<Integer,Strings> h = new hashmap<Integer,Strings>
        h.put(1, "Prabhash");
        h.put(2, "Pragya");
        h.put(3, "Prachi");
        h.put(4, "Osas");
Hashmap<Integer,String> h2 = new hashmap<Integer,String>
        h2.put(5, "Papa");
        h2.put(6, "mammy");
    sop(h.putAll(h2));
        Sop(h.size());
//      ~~sop~~ h.clear();
true. //  sop(h.isempty();
        Sop(h.get(1));
        ~~Sop~~ for(Entry<Integer,String> E : h.entrySet()){
            Sop(E.getKey()+":"+ E.getvalue();
        }
        Sop(containsKey(2));
        Sop(containsvalue("Pragya"); (we can directly print)
        Sop(remove(1));  ↓ it will give a empty
        map<Integer,String> m = Collections.Emptymap()
        for(Entry<Integer,String> E1 : m.entrySet()){
            Sop(E1.getKey()+":"+ E1.getvalue);
        }
        Sop(m.size());
  }
}
```

in entry interface. h.entrySet() returns a set of key & value pair that acts as an object. that provides implementation for Entry interface. So it is inherited but those method same so we will override in so we will get over ride implementation

(collection.emptymap will return an empty & immutable map) we are storing it in a map so that whenever we need empty implementation the empty map is stored in it.

O/P {1=Prabhash, 2=Pragya, 3=Prachi, 4=Osas, 5=Papa, 6=mummy}

3

6

Prabhash

1: Prabhash
2: Pragya
3: Prachi

4: Osas
5: Papa
6: mummy

True.
True.
{2:Pragya, ----}

0

sop(collections.emptymap) // o/p={}.
sop(h.entryset)//[1=Prabhas, ---- 500n];

papergrid

Date: / /

~~TREE M~~ LINKED HASH MAP

- Linked Hashmap ~~bele~~ is a predefined class belongs to Java.util package
- Linked Hashmap is a subclass of Map interface
- Linked Hashmap is by default generics.
- In Linked Hashmap size is dynamic.
- We cannot add duplicate keys in linked hashmap but we can add duplicate values.
- We can add both key & value as null.
- It follows order of insertion
- all the methods are non synchronized.

```
import Java.util.Linked hashMap.
Public class sample1{
 psvm
LinkedHashMap<String,Long> h= new Linked HashMap<
        h.put("sheela", 78964l);          String, Long>();
        h.put("susheela", 896543l);
        sop(h);
Linked HashMap ~~("mala",789654l)~~ <String, Long> h1=
new Linked hashmap <String, Long>();
        h1.put("mala",789654l);
        h1.put("Parimala", 789543l);
        sop(h1)
    } }  sop(h.putAll(h1));
O/p={sheela = 78964, susheela = 896543}.
    {mala = 789654l, Parimala=789543}
{sheela=78964, susheela=896543, mala=789654,
        Parimala=789543}.
```

TREE MAP

- Tree map is a ~~pre~~ predefined class of ~~map~~ Java.util package
- Tree map is a subclass of map interface
- In Tree map size is dynamic.
- Tree map is by default generics.

- treemap is completly autosorted & sorting is done on basis of key
- we cannot add null keys but we can set null values.
- we cannot add duplicate keys but we can add duplicate values
- It doesnot follows order of insertion.
- All the methods are not synchronized.

```
import java.util.mapentry;
import java.util.treemap;
class sample10{
  Psum(str[]args){
Treemap<integer, double> T= new treemap();
  T.put(15,2.25);
  T.put(6, 23.63);
  T.put(1, 8.25);
  T.put(5.6.36);
for(Entry<integer,double> E1 =T.entryset())
{
  sop(E1.getkey()+" " +e1.getvalue());
}
}
```

```
o/p - {1=8.25, 5=6.36, 15=2.25, 16=23.63}
      1    8.25
      5    6.36
      15   2.25
      16   23.63
```

```
import java.util. linkedlist
class sample{
  Psum(str[]args){
Linkedhashmap<character, float> l=
  new linkedhashmap();
  l.put('a', 2.25f)
  l.put('q', 5.25f);
  l.put('w', 5.88f);
  sop(l);
  sop(l.containskey('q'));
  sop(l.containsvalue(2.25f));
  l.remove('w');
  l.get('a');
}
}
o/p = {a=2.25, q=5.25, w=5.88}
true
true.
{a=2.25, q=5.25}
2.25
```

Garbage collection

In other programming language we have to call garbage collection explicity in Java JVM calls the garbage collection i.e finalize() implicitly.

- Whenever the referenced variable is dereferenced to null automatically finalize method will be called and finalize method will remove the unused memory from the heap memory.
- to call finalize method we have to call gc method of system class it is implicity called by JVM.

```
Public class sample extends
  Public void finalize();
    sop ("I am finaliz
  } Psum(str[]arg
    sample S= new
      S = null;
      System.g
    }
  }   op = I a
```

Note - maps are mainly used fo
on corresponding keys.

- Hashmap will not follo
- linked hashmap will foll
- Tree map is completely

we can create frontend
package.

Marker interface - An i
method is called as ma

Types of dupli

1) shallow copy - in case of
object then other obje

```
- Class employee{
    int id= 1;
    Str name="raju";
  Psum(str[]args){
  Employee E1= new employe
    sop(E1.Id);
    sop(E1.name);
  Employee E2 = E1;
    sop(E2.ID);
    sop(E2.name);
    e2.name="Suresh";
```

```
Public class sample extends object(){
    public void finalize(){
        Sop ("I am finalize()");
    } psvm(str [] args){
        sample S= new sample();
            S = null;
            System.gc();
        }
    }        op = I am finalize.
```

Note - maps are mainly used for quick retrival of data based on corresponding keys.

- Hashmap will not follow order of insertion
- linked hashmap will follow order of insertion
- Tree map is completely autosorted based on keys.

we can create frontend using Java by using awt package.

Marker interface - An interface that doesn't have any method is called as marked interface.

Types of duplicating the objects

1) Shallow copy - In case of shallow copy if we anyone object then other objects will get effected.

```
- Class employee{
    int id= 1;
    Str name= "raju";
  psvm(Str [] args){
  Employee E1= new employee();
    Sop(E1.Id);
    Sop(E1.name);
  Employee E2 = E1;
    Sop(E2.ID);
    Sop(E2.name);
    e2.name="Suresh";
    Sop(e1.name);
    Sop(e2.name);
  }
}
```

```
o/p-  1
      raju
      1
      raju
      Suresh
      Suresh.
```

Scanned with OKEN Scanner

② **deep copy** - If we modify any one object the other object will not get affected. but no of lines will get increased.

```
- Class Employee{
    ~~Psvm(Str~~ int id=1
                string name="raju";
    psvm (Str [] args){
      Employee E1 = new Employee();
        Sop(E1.id);
        Sop(E1.name);
      Employee E2 = new Employee();
          e2.id = e1.id;
          e2.name = e1.name;
      Sop (e2.id);
      Sop(e2.name);
      e2.name = "suresh";
      sop (e1.name);
      Sop(e2.name);
    }
}
```

o/p
1
raju
1
raju
raju
suresh

③ **clone()** - clone is an not static no final method of object class.

- the return type is object.
- clone is a combination of shallow copy & Deep copy
- It looks like a shallow copy but it behaves like a deep copy.

```
Class employee implements clonable{
  int id;
  String name;
  Psvm (Str [] args) throws clone Not supported Exception
    Employee E1 = new employee()
      e1.id = 1;
      e1.name = "raju";
    Employee E2 = (Employee) E1.clone() = return an object for
    sop (e1.id);                this class but of object type
    sop (e1.name);              so we need to downcast
                                object o = new employee()
                                return o
```

Scanned with OKEN Scanner

& is being provided to object class reference variable as an input & then being returned

papergrid

object obj = new ~~employee~~ object() employee
return obj;

as object class is extended to all the class so all the properties of the subclass in it. [Upcasted]

we are doing downcasting to convert object class to employee class object [it is upcasted]

```
sop(e2.Id)
sop(e2.name).
e2.name = "suresh";
sop(e1.name);
sop(e2.name);
}
}
```

Clonnable is a marker interface and Java is a secure language so it doesn't allows to clone the object directly for that we implement clonnable interface.

O/P = 1 raju
raju
raju
raju
suresh.

On Java Stack follows LIFO remaining all follows fifo

Stack

- Stack is a subclass of vector class
- stack follows LIFO
- On stack we have two extra methods Peek & Pop. (remaining points of vector).

```
import Java.util.stack
Public class mainclass{
PSVM(str[] args){
Stack S = new stack();
S.add(25);
S.add(30);
S.add(255);
Sop(S.peek());
Sop(S);
Sop("-------");
Stack S2 = new stack();
S2.add(25);
S2.add(30);
S2.add(255);
Sop(S2.pop());
Sop(S2);
}
}
```

Peek - It will fetch the Last element but won't delete it

Pop - It will fetch the ~~first~~ last element but it will delete it

O/P - 255
[25, 30, 255]
255
[25, 30];

Scanned with OKEN Scanner

utility classes

- collections
  - methods
  - sort
- Arrays
  - methods
  - sort.

```
import Java.util.Arraylist;
import Java.util.Arrays;
import Java.util.collections;
public class mainclass{
ArrayList a1 = new Arraylist();
    a.add(15);
    a.add(20);
    a.add(26);
    a.add(18);
    a.add(1);
Sop("beforting ->" + a);
    collections.sort(a);
Sop("After sorting->"+a);
Sop("----------");
int[]arr={1,5,8,2,6,7,8}
Sop("before sorting->" + Arrays.tostring(arr));
    Arrays.Sort(arr);
Sop("After Sorting->"+Arrays.tostring(arr));
}
}.
```

o/p – before sorting -> [15, 20, 26, 18, 1]
after sorting -> [1, 15, 18, 20, 26]
before sorting -> [1, 5, 8, 2, 6, 7, 8]
After sorting -> [1, 2, 5, 6, 8, 7, 8]