Page No.

Date: 27/09/2022
~~Wednesday~~
Tuesday

Name :- Nagaraja Shivappa Aralikalli

Sub :- Java ~~Full stack Development~~

Branch :- Jspiders Rajajinagar Bengaluru.

Mob No: 6361153251

Email Id: nagarajsaralikalli2022@gmail.com.

18/10

20/10

22/10

21/10

Presentation done 09/12

Presentation done 09/12

JSPIDERS JSP Presentation Sub: Date: RAJAJINAGAR

JSPIDERS 14-10 BENGALURU RAJAJINAGAR

Java ~~Softwares~~ Required software

1] Editors :-

- Notepad
- Edit ++

2] Command prompt

3] Java (Java Development Kit) Jdk 1.8 V

4] IDE (Eclipse, Netbeans) SE, ME, EE.

5] Java commands

- cls → clear screen.
- Mkdir → make directories.
- cd → change directory.
- cd.. → switch to other directory.
- Javac → Java compilier.
- Java → Java Interpreter.

6] New software Eclipse platform :

steps to download

- go to browser whatever you want like google, chrome, fire fox.
- Then enter name download eclipse neon3 for windows 10. then click first link.
- after Home page click eclipse (EE) Enterprise Edition.
- click on download button. then start download eclipse software. after download double click on that file & install it.

INDEX

SI.NO	CONCEPT'S	MOCK CONDUCT
1	Introduction to JAVA Language.	
2	software Initallation of Java Idle 1.8.0	
3	Java Architecture and software Development life cycle model.	
4	VARIABLES (*) classification of variables. (*) variable data types.	
5	METHODS (*) static methods. (*) Non-static methods. (*) Relence variable.	
6.	CONSTRUCTOR (*) simple construction (*) this. keyword constructor.	
7.	COMPOSITION	
8	PASS BY VALUE AND PASS BY REFERENCE.	
9	CLASS AND OBJECT	1st MOCK

SI.NO	CONCEPTS	MOCK CONDUCT
	OOP'S	
1	INHERITANCE (a) Types of inheritance.	
2	BLOCK STATEMENTS (a) SIB (b) IIB	
3	METHOD OVERLOADING	
4	METHOD OVERRIDING	2nd mock
5	TYPE CASTING	
6	POLYMORPHISM	
7	ABSTRACT CLASS	
8	INTERFACE	
9	PACKAGE	Pre - Final mock
10	ACCESS SPECIFIER	
11	ENCAPSULATION	
12	THREADS	
13	AGILE HANDLING	Final mock

INDEX

SI.NO	CONCEPTS	MOCK CONDUCT
	Java programming concepts	
1	LOOPING statements (*) for loop (*) while loop (*) do-while loop.	
2	CONDITION STATEMENT'S (*) if (*) if-else (*) if-else if (*) Nested if-else (*) switch case.	
3	Arrays	
4	programming (*) swap programs (*) Factorial program (*) prime Numbers (*) Even and odd program. (*) patterns (*) Fibonacci series (*) Reverse strings.	
5	STRINGS FUNCTIONS	1st mock

01/05/

Java

Basic concept's :

Computer (Common Orient Machine project used for Technical Education Research.)

"Computer is a electronic machine that contains software and Hardware parts, whats user Enter the inputs and ~~com~~ machine will provides Results."

Inputs Devices in computer :-

1] Keyboard → keyboard is a device that use Enter keys.
2] Mouse → mouse is a Devices user, use that device insid.
3] CPU → Central processing Unit.
4] Scanner → Scanner is Device scan the html
5] microphone → Devices
6] Touch screen → Devices that touch that command executes.

Output Devices in Computer :-

1] Monitor → monitor is a Device, when user see result in Display.
2] Pen drive → outside connector.
3] CD Drive → compact Disk outside Device.
4] printer → printer is a o/p device to print note.
5] projector → projector is a o/p device to display screen.
6] Head phones → Head phone is a o/p device to listen outside.
7] Computer speacker → o/p device speakers.

Languages :-

1] Communication languages → Humans

2] Computer languages → machine.

1] Communication language :-

Communication language is a Humans languages To communicate one or more peoples to talk using Humans languages. ~~Thear~~ There are.

1] Kannada
2] English
3] Hindi
4] Hrudu.
5] Telagu
6] Tamil.
7] malayalam.
8] France
9] Japanies.
10] ~~Chines~~ chainise. etc...

and many more ~~to~~ Human languages in this world.

* Every Human languages has culture and rules & regul-ations are there, follow those, to learn and talk. To knows every languages.

* Don't compare one language to another langugе. because every language has there culture rules & regulations are there. If you compared you don't understant language.

* follow the every language culture rules and regulation to eaiy understood.

This all about Humans languages.

2] Computer languages :- Computer language is a medium, communicate between people to machine. The computer languages are following bellow :-

(*) There are over 300 programming languages in the world.

(*) The computer languages are:

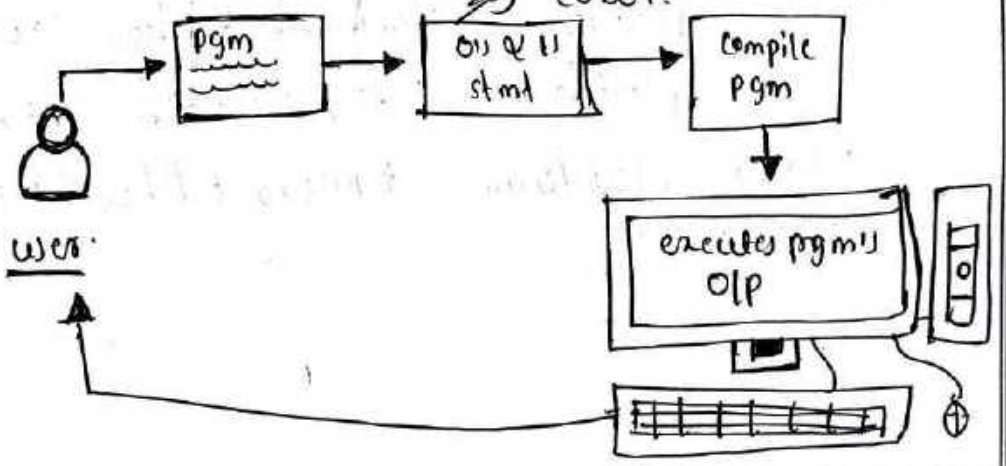
1] C
2] C++, C#
3] Python
4] Java
5] SQL
6] swift
7] Ruby
8] kotlin
9] perl
10] MATLAB
11] Dart
12] scala.
13] web Technologies
- (*) HTML
- (*) CSS
- (*) Javascript

} front end

- (*) php
- (*) SQL

} Backend

14] Flutter.
15] Assembly language.
16] objective C
17] Lisp.
18] visual basic
19] oops
20] cobol.



JAVA [redacted]

Java Father: "James gosling"
"1995"

Introduction To Java :-

- Java is a computer programming language.
- Java is High level, class based, object oriented programming language.

Java used for :-

Java programming used for, Developers use Java to construct applications in laptops, data centers, game consoles, scientific super computers, cell phones and other devices.

There are two types of Java programs

1) Java stand-Alone Applications.
2) Java applets.

There are four platforms of the Java programming

1) Java platform, standard Edition (SE)
2) Java platform, Enterprise Edition (EE)
3) Java platform, Micro Edition (ME).
4) Java FX.

Java Concepts :-

A] TOKENS :- Tokens are the smallest unit of Java program.

In Tokens some of following bellow :-

- Identifiers → To identifies the name.
- Key words → predefined words.
- Literals
- operators
- separators
- comments.

1] Identifiers :- Identifiers are the name given for Java programs.

2] Keywords :- Keywords are the predefined words which has own meaning in Java.

There are 50 keywords in Java.

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.

3) Literals :- Literals are the values in java program

1) Numerical literals :- The Numerical literals are the Numbers, Integer values.

10, 20, 30 . . . .

(*) Decimal values :- Decimal values are the Number within the Dot wise 10.2, 10.3, 10.4 . . .

2) Character literals :- The character literals have two type (*) Single characters (*) double characters.

(*) Single characters :- The single characters are the single quotes inside write anything that is single character.

for example :- 'A' 'B' '10' '10.2'

(*) double character / String literals :- The double character / string literals within the double quotes write Anything is called string literal.

for ex: " Hello ", " JAVA " " 10 " .

4] Operators :- operator are the symbols making operations/task.

1] Unary operator → { exp++ exp-- postfix
++exp --exp prefix

2] Arithmatic operator → + - X /

3] Relational operator → > < >= <= == !=

4] Logical operators → ! && || (NOT AND OR) =

5] shift operator → > < ^ >

5] Separators :- separator is the process of making separate section / ~~separates Department~~ that have symbols.

{ } → braces

[ ] → brackets

( ) → parantheses

, → coma

; → semicolen.

6] Comments :- comments is the comments the code, that gives long information about code. line by line.

// Single comment.

/* - - -

- - - */

Software Development life cycle Model (SDLCM)

Software :- software is a collection of programs / software is set of computer programs and associate domentation data.

SDLCM :- software Development life cycle is a step by step procedure / standard procedure to build an new software.

software stages :-

1] Requirement collection :- Requirement collection is the 1st stage of SDLC Model. in this Requirement collection, There is a one person called BA Business Analyst. any software company there is a one person is called BA (Business Analyst). The BA will collects the Requirements who wants software for their Business. The BA will collects their complete information, after collecting the Requirement that converts into customer Requirement specification into software Requirement specification (CRS - SRS).

2] Feasibility study :- Feasibility study is the 2nd stage of SDLC model.

Feasibility study have There is a member

1. Project Manager (PM)
2. Human Resources (HR)
3. Finance Team (FT)
4. Developers Team (DT)
5. Testers Team (TT)

The above Teams are checks the projects status and whats new project is comming & Track the project.

1. PM is Handling & manage the projects.
2. HR Team will check availability is the Resources.
3. Finanace Team make Financial Reports.
4. Developers is build softwares.
5. Tester is Tests the softwares.

3) Design: Design is the SDLC ~~the~~ Model 3rd stage To make Designs (Architecture)

1. ~~Low~~ High Level Design (HLD) External
2. Low level Design (LLD) Internal.

4) Coding: coding is a process of writing programs in Editors and executes.

5) Testing: Test the build slows.

6) Installation: Installing softwares in Internet.

7) Implementing: Implements the softwares.

8) Maintainance: managing software / secure.

SDLC Model Diagram :

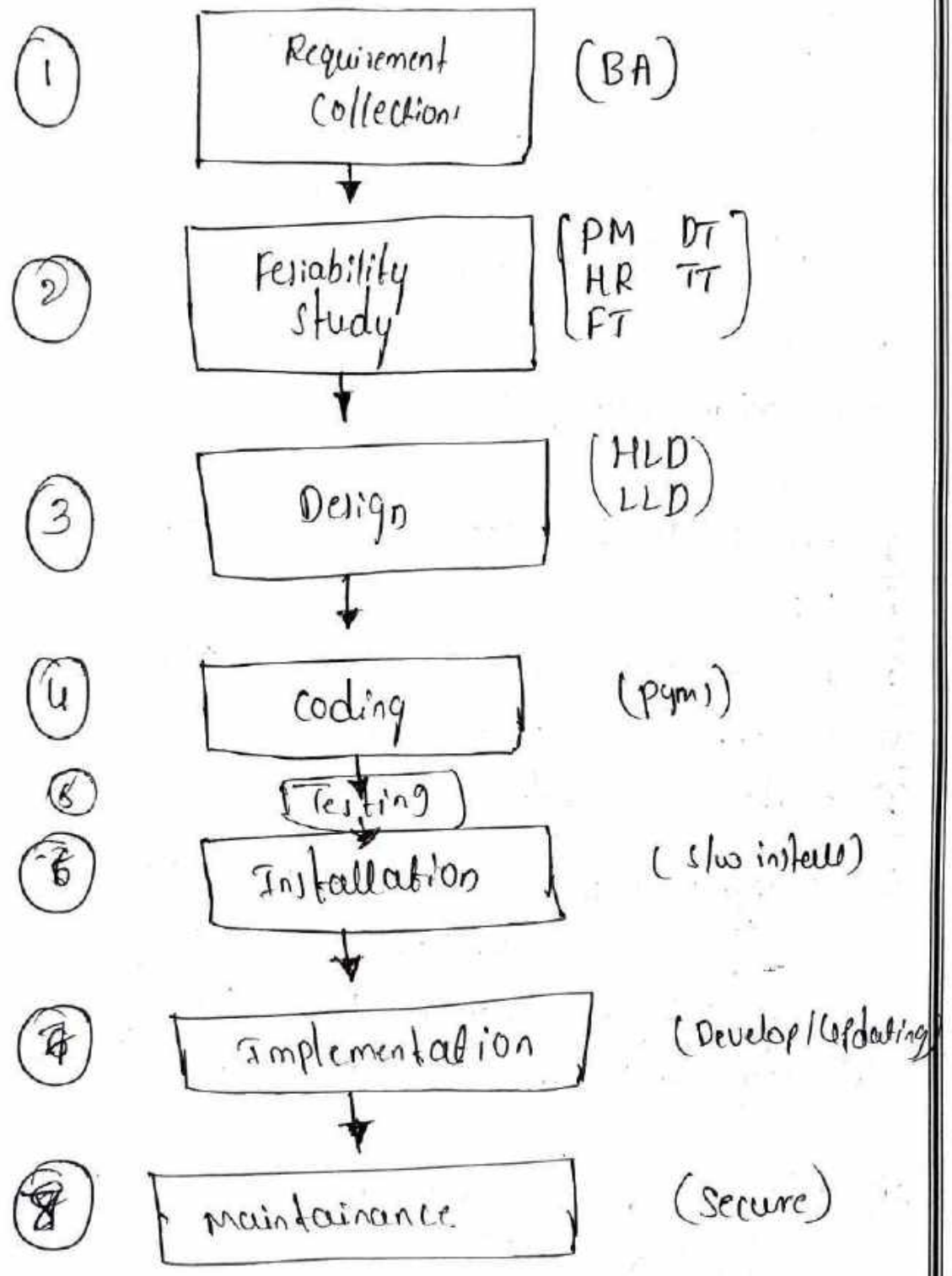
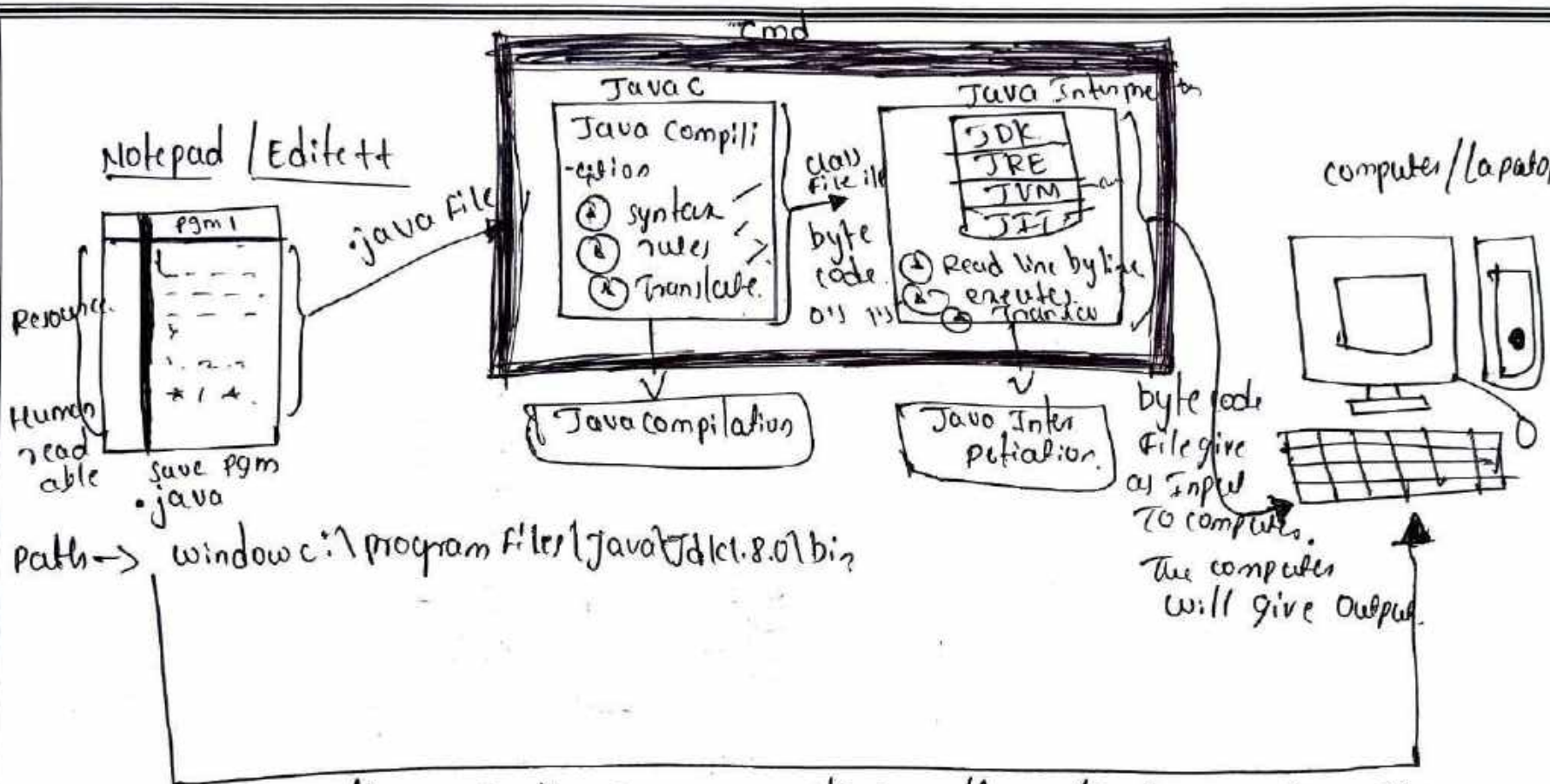


fig: SDLC.

JVM :- Java virtual Machine, it is use to run the programs by other language, also what compiler compiles the code & convert into class ~~code~~ file.

Java Architecture :- Java Architecture is a process of how we write program & execute.



after wrote the pgm we will see the o/p in computer. But the computer will understand only machine language. 0's / 1's we wrote Human readable language. so we need command prompt. This is the execution Area.

JRE :- Java Runtime Environment it is a slow, it ~~requires~~ Java pgms require to run correctly.

JDK :- it Java Development Kit which is without this tool we can run the Java programs

~~Simple java program~~ :

JAVA TYPES :

1] class
2] enum
3] Interface.
4] annotation.

1] class :- class is a java Type. and it is keyword.

simple java program using class :

```
class Simple // class Declaration
{ keyword identifier
  start point public static void main (String[] args)
  {
    System.out.println ("Hello world");
    printing stmt
  }
}
```

class Body

inside body

Output

Hello world

Program 1 :- Write a java program to print Literals

" Hi " + 2 → Hi2

Concatination
Two Join Two Stmt's

```
Class Program1
{
    Public static void main (String[] args)
    {
      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 " + (20+20));
      System.out.println("The value is "+ 1/2);
      System.out.println("The value is "+ 1/2.0);
    }
}
```

+

2020

Output :-

Window c:\program Files\Java\jdk1.8.0\bin Path.

c:\program files\Java\jdk1.8.0\bin> javac Program1.java → Compilation

c:\program files\Java\jdk 1.8.0\bin> java program1 → run.

40
~~2020~~ The value is 2020
40 is The value.
40 The value.
The value is 0
The value is 0.5

6/10/2022

Variables :-

"Variables is a named memory location, which can store some values / Data, & it can change 'n' number of times during execution."

Variables are two types :-

1] Primitive Data type
2] Non-primitive Data type / Refere type / class type.

1] Primitive - Data type :-

- byte
- short
- int
- long
- float
- double
- Char
- boolean.

2] Non - Primitive Data type

- Arrays
- Strings
- or
- Any class type.

variable Syntax :-

1] Variable Declaration :-

Syntax: datatype variable name ;

ex : int a ;

2) Variable Initilization :-

syntex : variable name = value;

ex ~~syntax~~ : a = 10 ;

3) Variable utilization :-

syntax : System.out.println (~~"value"~~ variable name);

ex: System.out.println (a);

4) Variable Re-Initilization :-

syntax: data type variable name = value;

variable name = ~~value;~~ other value;

ex: int a = 10;

a = 20;

5) copying the value from one variable to another variable :-

syntax: data type variable name = value;

data type new variable name = ~~change~~ same value;

ex: int a = 10;

int b = a;

SOP (a);

sop (b);

10
10

Simple variable program :-

```
class simple
{
    public static void main (String[] args)
    {
        int a; //variable Dellaration
        a= 10; //variable initilization
        System.out.println(a); // variable utilization.
        System.out.println("The value is"+a);
        System.out.println(20+a+20);
    }
}
```

Output

```
c:\program files\java\jdk 1.8.0\bin>javacsimple.java
c:\program files\Java\jdk 1.8.0\bin>java simple.
10
The value is 10.
50
```

Re-initilization program :

```
class simple1
{
  public static void main(String[] args)
  {
    int a = 10;
        a = 20;
        a = 30;
    system.out.println("a");
    system.out.println(30+a);
  }
}

}
```

Output

```
c:\program files\Java\Jdk1.8.0\bin> Javac simple1.java
c:\program files\Java\Jdk1.8.0\bin> Java simple1

30
60
```

Arithmatic's operation :

```
class Sample 2
{
  public static void main(String [] args)
  {
    int a ;
    int b ;

    a = 10 ;
    b = 10 ;
    c = a+b
    System.out.println( c);
  }
}
```

20

```
class Sample 3
{
  public static void main (String [] args)
  {
    int a = 10;
    int b = 90;
    S.O.P ("The value is" + a + b);
  }
}
```

The value is 1090

Arithamatical operations :-

```
class program2
{
  Public static void main (String [] args)
  {
    int a = 100;
    int b = 200;
    int c = a+b;
    System.out.println("The Addition of two No's:"+ c);
  }

    int a = 100;
    int b = 200;
    int c = a-b;
    System.out.println("The substraction of two No's:"+ c);

    int a = 100;
    int b = 200;
    int c = a*b;
    System.out.println("The multiplication of two No's"+ c)

    int a = 100;
    int b = 200;
    int c = a/b;
    System.out.println("The Division of two No's"+ c);
  }
}
```

7/10/2022

Classifications of variables :-

1] Local variables
2] Global variables

Local variables	Global variables.
Any variables which is Declared within the method. it is called as Local variable:	Any variables which is Declared ~~within~~ outside method inside in class is called as Global variable.
The scope of local variable is from beginning of the method to End of the method.	The scope of the variable is from beginning of the class to End of the class.
Local variables doesn't have the default values.	Global variables has a default values.
Local variables can'nt be classified as static & non static.	Global variables ~~has~~ can be classified has static non-static.
~~Local~~ variables after Declaring the variables	Global variables after declaring the variables

- immediately in a next line we can reinitilization a variables.

- immediately in the next line we cannot reinitilize the variable.

Ex:

Local variable pgm :-

```
class Demo
{
  public static void main (String[] args)
  {
    int a=50;
    System.out.println(a);
  }
}
```

Global variable pgm :

```
class Demo
{
static int a=50;
  public static void main (String[] args)
  {
    System.out.println(a);
  }
}
```

default value of Data types :-

1) byte → 0
2) short → 0
3) int → 0
4) long → 0
5) float → 0.0f
6) double → 0.0d
7) boolean → false.
8) Char → '0' ic 1000 (code)

keywords datatype

9) String → null

This is not keyword

Note:

① we can't Declare a variable outside the class

② we can't use a print statement outside the method

8/10/2022

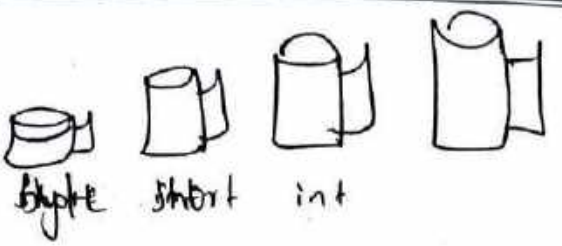
Primitive Data Types Size & Ranges :-

The Difference b/w the primitive Data types are size

$$\boxed{-2^{n-1} \text{ to } 2^{n-1}-1}$$

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

Mimitive Data Types



~~Range~~

byte → 0
short → 0
int → 0
long → 0

} Integer type of Data

float → 0.0f

double → 0.0d

char → "Unicode"

boolean → 1 true false.

byte →

Real time examples on variables :-

Rajesh joined Jspiders by paying fees 35,000 He started with the subject Java and He belongs to 'A' group.

Write the program to print Rajesh Details.

```
class Static
{
static String stud_name = "Rajesh";
static int    fees = 35000;
static char   group = 'A';
    public static void main (String[] args)
    {
        System.out.println(" The student name is "+ stud_name);
        System.out.println(" The fees of student is "+ fees);
        System.out.println(" The group name is "+group);

    }

}
```

Suresh Joined IBM His Empoyee Id is 123, Dept Name is 'B' ~~The~~ His salary is 56000.56.

Write a program to Display Employee Details.

```
class Employee
{
static String Emp_name = "Suresh";
static String Emp_Company_name = "IBM";
static int Emp_Id = "123";
static float salary = "56000.56";
static char Dept_name = 'B';
public static void main (String[] args)
{
System.out.println("The Emp_name is:"+ Emp_name);
System.out.println("The Emp CompanyName is:"+ Emp_Company_name);
System.out.Println("The Employee Id is"+ Emp_Id);
System.out.Println("The Employee salary is "+ salary);

}

}
```

11/10/2022

Conditional statements :-

(R) Whenever programmer ~~once~~ wants to check a logical condition, ~~when~~ then we will go for conditional statement

Types of conditional statements :-

1) if
2) if-else
3) else-if ladder
4) nested if-else.

Syntax :-

1) if :-

```
class Demo
{
    public static void main(String[] args)
    {
        if (condition)
        {
            //statement ;
        }
    }
}
```

example of if condition

```
if (10>8)
{s.o.p("10 is greater");
}
```

2) if-else statement :-

```
if (condition)
{
    //statements;
}
else
{
    //statement 2;
}
```

example of if-else condition

```
if (10<2)
{s.o.p("10 is not lesser");
}else
    s.o.p("2 is lesser");
}
```

3] Use-if statement:

```
class Demo
{
  Public static void main (String [] args)
  {
    if (condition1)
    {
      //statement;
    }
    else-if (condition2)
    {
      //statement;
    }
    else-if (condition3)
    {
      //statement;
    }
    :
    :
    else
    {
      //statement;
    }
  }
}
```

Example

```
if (100>1000) //condition1
{
  S.O.P("The 100 is not greater");
}
else if (2000>1000) //condition2
{
  S.O.P("The 2000 is greater");
}
else
{
  S.O.P("The in Numbers is greater");
}
```

4) nested if-else :- nested if else is a again recycle condition.

Syntax :-

```
if (condition)
{
    if (condition)
    {
        //statement;
    }
    else
    {
        // statement;
    }
}
else
{
    //statement;
}
```

example :-

```
if(8>100)
{
    if(100>8)
    {
        S.o.p("100 is greater");
    }
    else
    {
        S.o.p("The 100 is greater");
    }
}
else
{
    S.o.p("100 is greater");
}
```

METHODS :

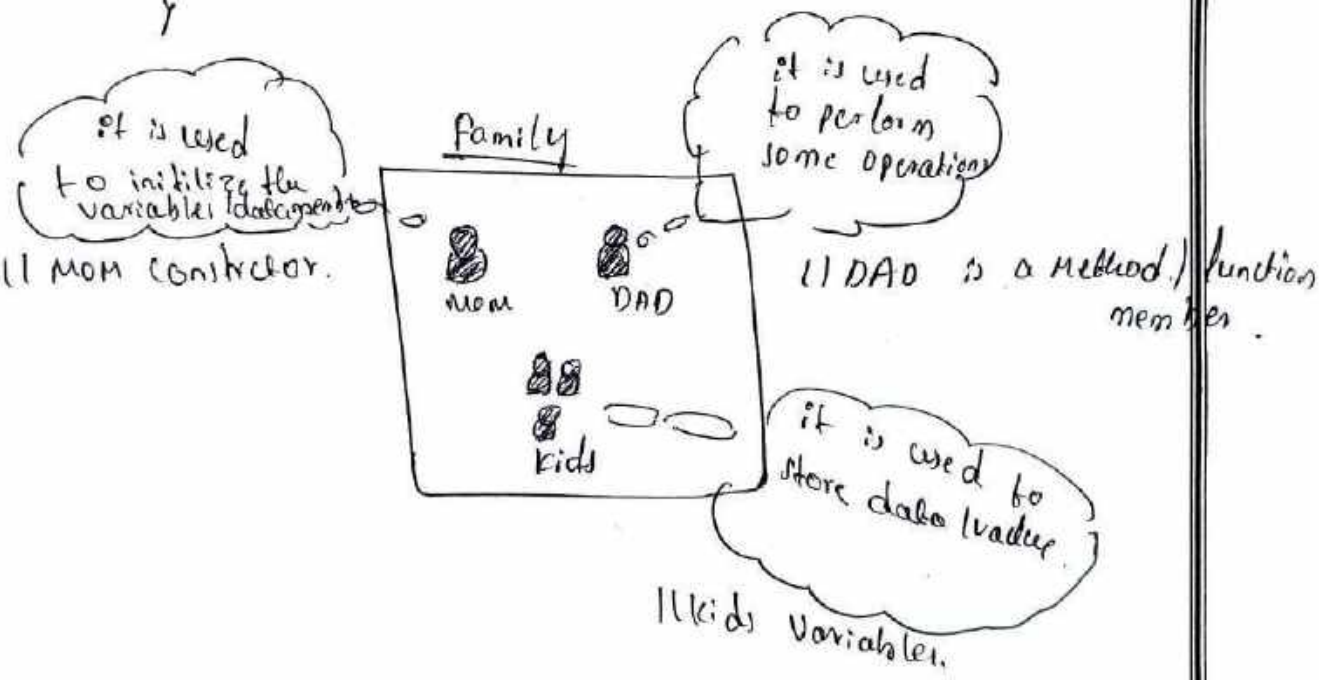
(?) In class we have three members, that is

1] Variable / Data member :- it is used store some value / Data.

2] Method / Function member :- it is used to perform some operation.

3] Constructor :- it is used to initilize variables - / Data members.

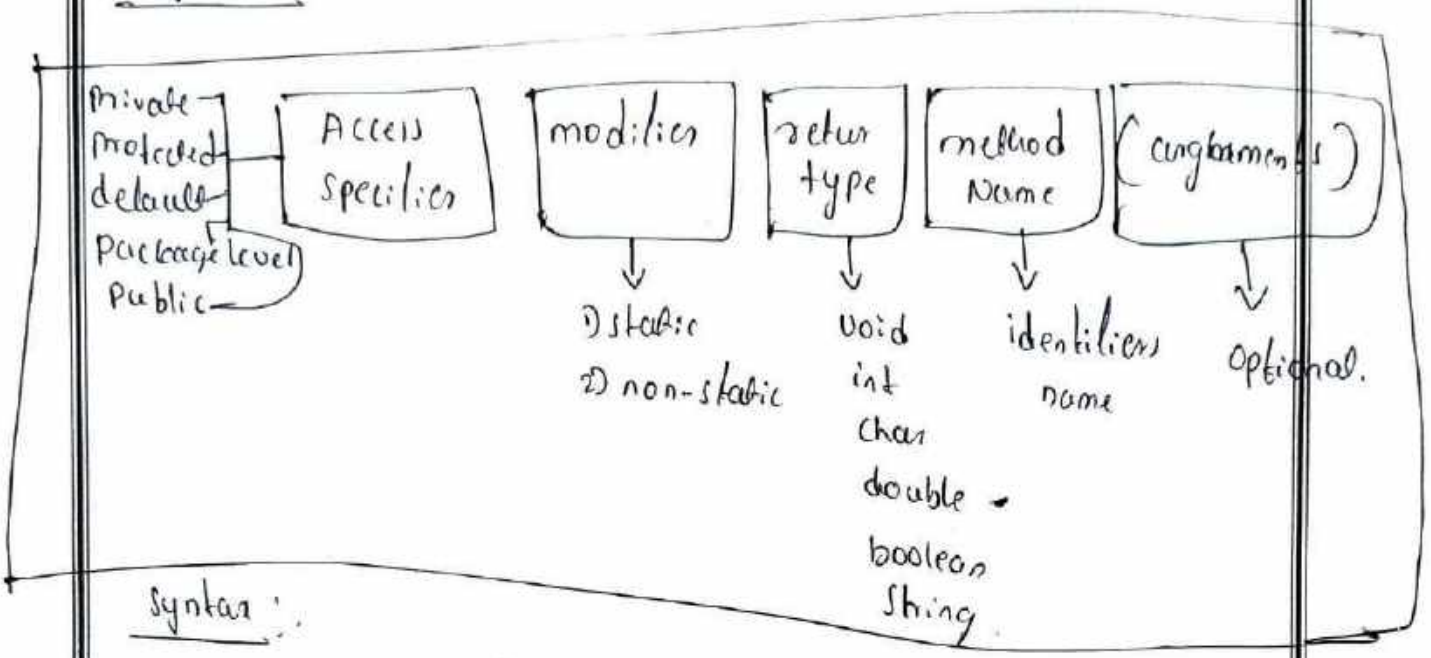
```
class   class_Name
{
static      variable / data member
non-static  Method / function member
            Constructor // it is always non-static.
}
```



Dingi
Dinga
1st Anniversary.
2nd Anniversary
3rd anr.

Method :- Method is a block of statment which will get executed, whenever it is called / invoke.

Syntax :-



Syntax :

access specifier modifier return type method name (argument)

```
{
    stmts ;

    return ;
}
```

method body definition

// return works it gives back what we return Data.

Public static void main (String [] args) // This is the starting point of Java program & public is access specifier, static is modifier void is return type main method name (String [] args) is argument optional..

For example :

```
class Demo
{
    static void add ()
    {
        int a = 10;
        int b = 20;
        int c = a+b;
        system.out.println (c);
    }

    public static void main (String [] args)
    {
        System.out.Println("Main - start");
        add();
        System.out.Println("main End");
    }
}
```

30

Write a program to multiply three Numbers using method.

```
class Sample
{
    static mul ( )
    {
        int a = 2;
        int b = 3;
        int c = 4;
        int res = a*b*c;
        S.o.println (res);
    }
    Public static void main (String [] args)
    {
        System.out.Println(" Main start");
        mul ();
        S.ystem.out.Println(" Main - End");
    }
}
```

Write a program to multiply four numbers using method.

```
class Mull
{
    Static void mul ( )
    {
      int a = 40;
      int b= 20;
      int c= 30;
      int d= 40;
      int rel= a*b*c*d;
      System.out.Println(rel);
    Public Static void main(String[] args)
    {
        System.out.println(" The multiplication of four No's");
         mul();
        System.out.Println(" Thank you for checking program");

    }

}
```

OUTPUT :

```
The multiplication of four No's
240,000
Thank you for checking program.
```

```
class Demo
{
    static void area ()
    {
      final double  Pi = 3.142
            int     r = 5;
      double res = Pi * r * r;
      System.out.println (res);
    } return ();
    Public static void main (String [] args)
    {
      System.out.println ("The Result is bellow");
        area();
      System.out.println ("Thankyou");
    }
}
```

OUTPUT

```
The Result is bellow
78.55
Thank you.
```

Real time Examples on Methods :-

Develop and application flow for phonepe, where it's has following features.

1) Sing up
2) Login
3) pay
4) Logout.

```
class Phonepe
{
    static void Sign()
    {
     System.out.println("Sign up");

    }
    static void Login()
    {
     System.out.Println("Login");
    }
    static void Pay()
     {
      System.out.Println("pay");
     }
     static void Logout()
     {System.out.Println("Logout");
    }
    public static void main(String[] args)
    {
      System.out.Println("Welcome Phone pe app");
       Sign();
       Login();
       pay();
       Logout();
    }
}
```

```
Welcome Phonepe app
 Signup.
 Login
 pay
 Logout
```

Write a program to add two program with method with parameters. and b/w classes.

```
class Sample
{
  satic void disp( int num1, int num2)
  {
    //int num1 ;
    //int num2;
  int res num1+num2;
    s.o.pln ("The num1 and num2 adding "+ res);
  }
}

class sample_temp
{
  satic void display ( int a, int b)
  {
    int res = a+b ;
    s.o.pln ("The adding two values is "+ res);
  }
}

class Main class
{
  public static void main (String[] args)
  {
    sample.disp (10, 20);
    sample_temp.display (1, 2);
  }
}
```

METHOD'S using parameters :

13/10/2022

Write a program to calculate the total bill based on the quantity assign input.

~~Parameter~~ :

```
class Demo
{
    static void calc(int qnty)
    {
        int price = 10;
        int c = qnty * price;
        System.out.println("The calculation is :" + c);
    }
    public static void main(String [] args)
    {
        System.out.println(" This is calculation using parameter
                                passing in arguments");
        calc(2);
        calc(8);
        System.out.println(" Thank you");
    }
}
```

OUTPUT

```
This is calculation using parameter passing in arguments
20
80
Thank you.
```

Write a program to multiply Three number with method with parameter.

```
Class Mul
{
    static void mul ( int a, int b, int c)
    {
        c = a*b*c;
        System.out.println(" The multiplication of Three Nos "+ c);

    }

    public static void main (String[] args)
    {
        System.out.println (" The multiplication of Three nos");
        mul (2, 3, 2);
        System.out.println (" Thank you");

    }

}
```

Write a program to calculate Area of circle. with method with parameters.

```
class Pai
{
    static void Pai ( int r )
    {
        final double Pi = 3.142;
            double res = Pi*r*r;
        System.out.println (" The circle is"+ res)
    }

    Public static void main (String [] args)
    {
        System.out.println (" The Area of circle is bellow");
        pai (2);
        System.out.println (" Thank you");
    }
}
```

Looping statements :

(*) Whenever we have perform any operation repetedly. 'n' numbers of times.

in loop we have :

1] for loop
2] while loop
3] do while loop.

1] for loop : When ever we have to perform same operation repeatedly 'n' number of times. Then we go for for loop.

(*) Whenever we know the starting + Ending range then we go for for loop.

SYNTAX :

```
for (datatype variable initilization ; condition ; increment/decrement)
    {

    }
```

Example: Write Java program to Print 1 - 10 No's

```
class For
{

    public static void main(String[] args)
                    2<=10   2++
                    1<=10   1++
    {
        for( int i=1 ; i<=10 ; i++)
            {

                system.out.println("The 1-10 Nos Disply" + i);

            }

    }

}
```

1
2
.
10

14/10/2022

Write a program to add two numbers.

```
class Sample
{
    static int add ( )
    {
        int a = 10;
        int b = 20;
        int c = a+b;
        return c;
    }

    public static void main (String [] args)
    {
        System.out.println ("This method using return value");
        int x = add ( );
        System.out.println ("The addition of two no is"+x);
    }
}
```

JVM ①

② 20

executing cod ③

OUTPUT:

```
This method using return value.
The Addition of two no is: 30
```

Write a program Multiplication of Three numbers using Method return.

```
class Mul
{
    static int mul ( )
    {
        int a = 2;
        int b = 3;
        int c = 2;
        int res = a * b * c;
    }   return res;

    public static void main(String [] args)
    {
        System.out.println("The Multiplication of Three No's:");
            int x = mul ( );
        System.out.println("The Result is: " + x);
    }
}
```

OUTPUT:

```
The Multiplication of Three No's
The Result is: 8
```

Write a program to calculate Area of circle with method with return type.

```
class Area
{
  static double area( )
  {
    final double pi = 3.142;
        int  r = 5;
      double res = pi*r*r;
      return res;
  }

  public static void main (String [] args)
  {
      system.out.println ("The Area of circle bellow");
        double x = area()
      system.out.println ("The Result is: x);
  }

}
```

OUTPUT:

```
The Area of circle is bellow
The Result is: 78.55.
```

Real time example on method with return type

1) Janavi ordered Dairy-Mille silk in swiggy. the cost of the dairy mille is 100 Rupees. where she wants that chocolate very urgently, when the swiggy app showing 30 minutes to get delivered. but delivery boy came in 5 mins she was excited rather than paying 100 rupees she gave 120 rupees.

2) Write a program To display the Amount paid by Janavi.

```
class Sample
{
    static int swiggy()
    {
        int pay=100;
        return pay;
    }
    Public static void main(String[] args)
    {
        int x= swiggy();
        int tips=20
    }
        System.out.Println(x+tips);
}
```

Static: Any member declared to the keyword static is called as static ~~no~~ member of the class.

- static is always associated with class.
- static is one copy. All the static members will get stored in static pool area.
- Whenever we want to access static ~~mem~~ members from one class to another class. we have to use.

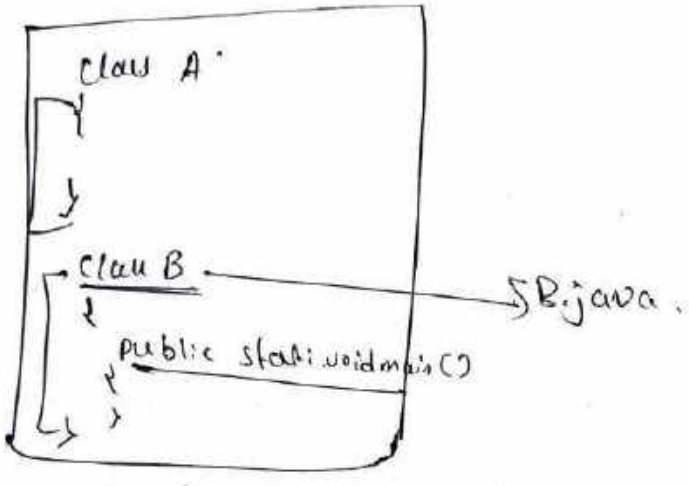
class name. variable name; or class name. method name();

Example:

```
class Sample
{
static  { Variable / data members
Non-static  { method (function members.
always ← constructor
nonstatic
}

class Demo
{
  static int a = 10;
  static void disp();
        {
        System.out.println("Hello");
        }
        Public static void main (String[] args)
        {
            System.out.println(a);
            disp();
        }
}
```

(*) In a ~~class~~ file we can develop multiple calss, which ever class having main method that class name should be the file name.

(*) If we have no number of class in a file those many .class file will get generated.



Ex:

```
class Sample
{
    int a=10;
    static void()
    {
        System.out.println("Hello+Nagarg");
    }
}

class Demo
{
public static void main(String[] args)
{
    System.out.println (Sample.a);
    Sample.void();
}
}
```

• Class programs :

17/10/2022

```
class Add
{
    static void add()
    {
        int a=10;
        int b=20;
        int c=a+b;
    }   System.out.println("The Add of two Nos is "+c);
}
class Mult
{
    static void mult()
    {   int a=2;
        int b=3;
        int c=a*b;
    }   System.out.println("The multiplication of three Nos "+c);
}
public static void main(String [args])
{
    Add.add();
    Mult.mult();
}
}
```

OUTPUT

```
The Add of two Nos 30
The multiplication of three Nos 6
```

Area of Circle :

```
class Area
{
 static void area( )
 {
   final double pi= 3.142;
    int r = 10;
    double res = pi*r*r;
 }  system.out. println (res);
}
class Sample
{
   static void sum( )
   {  int a = 100;
      int b= 200;
      int c = a+b;
}     system.out.println (c);
 public static void main (String[] args)

    Area. area( );
    Sample. sum( );

}
```

OUTPUT

3.142.2

3.142.2

Write a program to calculate Area of circle with method with parameter class.

```
class Area
{
  static void add(int a, int b)
  {
      int c = a+b;
      S.O.P(c);
  }
 class Para
 {
  static void area(int r)
  {
    final double pi = 3.142;
      double res = pi*r*r;
      S.O.P(res);

  }
  public static void main(String[] args)
  {
      Para.area(10);
      Area.add(10,20);

  }
}
```

```
class Return
{
  static int add ( )
  {
     int a = 10;
     int b = 20;
     int c = a+b;
  }  int res = a+b;
     return res;
}

class Type
  { static double area ()
    {
     final double pi = 3.1415;
      int r = 5;
     double res = pi*r*r;
     return res;
    }
    public static void main (String [] args)
    {
        Type ~~Type~~ double x = Type . area ();
                System.out. println (" The Area of circle is :" + x);

                int z ~~add Return ( )~~ = Return. add ();
                System. out. println (" The add of two no " + z);
    }
}
```

OUTPUT

```
78.55
30
```

Non-static :-

- Any member declared without the keyword static is called as non-static member 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 will create object that is

object . variable name	or	object. method name

object. reference . variable name

or

object. reference. method name

creating object :- to create object for accessing data in a class through this object.

SYNTAX:

new class_name ( );

ex: | new Tester ( ); |

```
class Tester
{
    int a = 10;
    void disp()
    {
        System.out.println("Hello Java");

    }

    public static void main(String[] args)
    {
        System.out.println(new Tester().a);
        new Tester().disp();
    }
}
```

NON - STATIC :-

18/10/2022

Badam

Pistal

Non-static Method

(1) Non-static is a Declared variable members / Declared method members without using static. (Non-static / Static) not fixed / fixed

(*) Non-static ~~object creation~~ variable members / Declared method members access the Data throug object.

(1) objects is the nothing but access the Data in a variable members / Declared method members with non-static only.

(2) SYNTAX of object creation :-

```
new classname ();
```

```
class Sample
{
    void add ()
    {

    }

    public static void main (String [] args)
    {

    }
}
```

Non-static Method :

```
class Sample
{
    void add()
    {
      int a=100;
      int b=200;
      int c=a+b;
      system.out.println(c);
    }

    public static void main(String[] args)
    {
        new Sample().add();
        new Sample().add();

    }
}
```

syntax for object creation :

$\underbrace{\text{new class_name ();}}_{\text{Object}}$

① object is the nothing but access data into the non-static members we used to create object. object through we will access it.

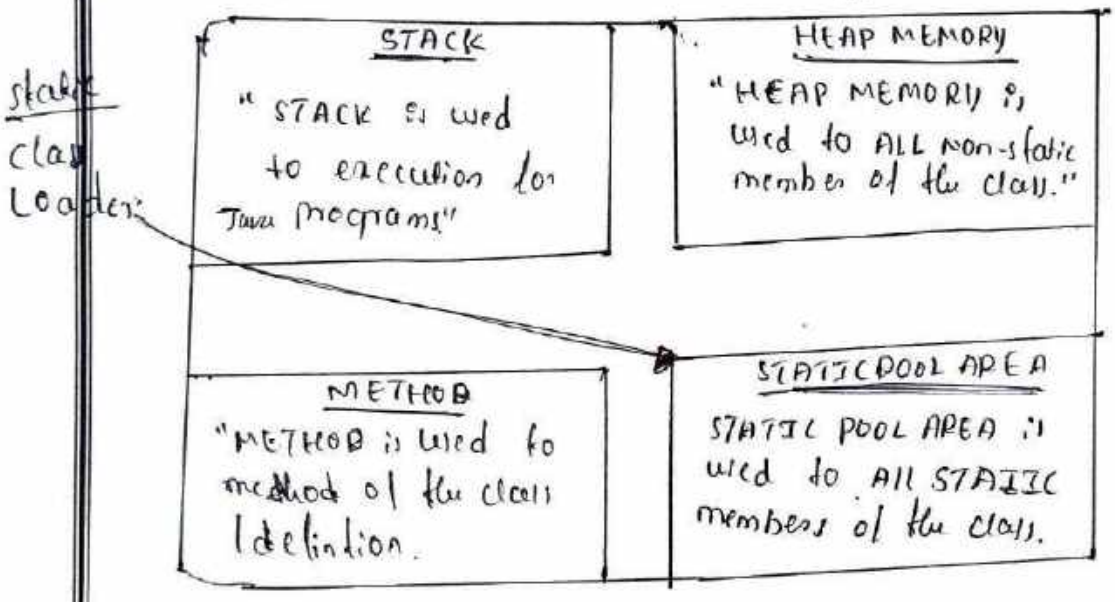
(A) new : new is the operator, it is used create a Random memory space into the Heap memory.

(B) Constructor : Constructor is used to initilize the non static members into the heap memory.

(E) RAM (Random Access Memory)

JVM (Java Virtual Machine

JVM has divided into 4 types



(I) class Loader : All the static method of members to load the class store in STATIC POOL AREA.

(K) method : in a class whatever methods are method member in the class store in a method.

(J) STACK : it's Brings the ~~Stat~~ static members and methods to executes the program.

(iii) Heap memory : stores the non-static or method members storage area;

19/10/2022

Non-static method :

1) Non-static method it does not contains the static method.
2) when we call the non-static method we will create object.
3) object creation. new classname(). methodname();

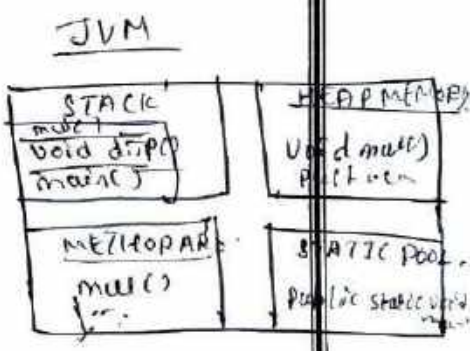
```
class Sampe
{
    void mul()
    {
      int a = 10;
      int b = 20;
      int c = a*b;
      System.out.println("The multiplication of Three no's:"c);
    }

    public static void main(String [] args)
    {
        new Sample(). mul();
        new Sample(). mul();
    }
}
```

o/p :

```
The multiplication of Three no's: 200)!
The multiplication of Three no's 200
```

Write a program to multiply three No's from non-static to static method.

```
class Non-static
{
    void mul()
    {
     int a= 10;
     int b= 20;
     int c = 30;
     int res = a*b*c;
     System.out.println("The multiplication of three No's is:"+res);
    }

    public static void main(String[] args)
    {
        new Nonstatic().mul();
        new Nonstatic().mul();
    }
}
```



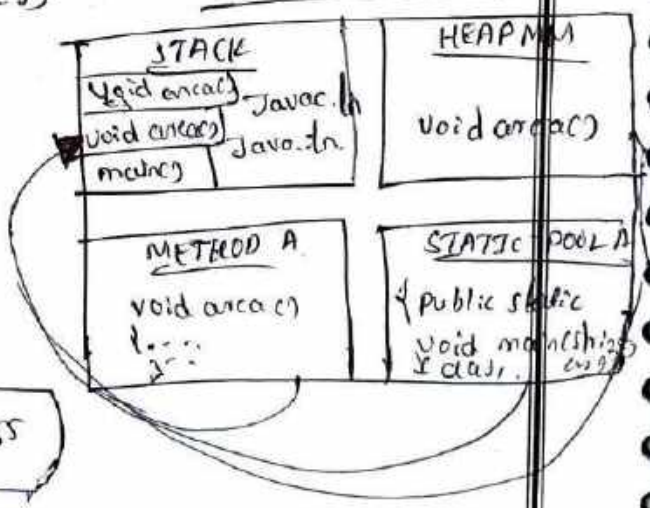
OUTPUT:

```
The multiplication of three No's is: 6000
The multiplication of three No's is: 6000
```

Write a ~~class~~ calculate Area of Circle from non-static to static method with JVM memory.

```
class Area
{
  void area( )
  {
    final double pi=3.142;
    int r = 5;
    double res = pi * r*r;
    System.out.println("The Area of circle is:"+res);


  public static void main(String[] args)
  {
      new Area( ). area( );
  }
}
```

JVM space



OUTPUT :

The Area of circle is: 78.55

20/10/2022.

REFERENCE VARIABLE :

- Reference variable is a special type of variable which is store object address.
- A reference variable can hold, either object address, / Null.
- reference variable is access Data in class, we need to create reference variables.

SYNTAX of Reference variables :

1] reference variable ~~Data~~ Declaration

syntax : `class_name reference_variable ;`

Ex : `Sample n2;`

2] reference variable Initilization :

syntax : `reference variable = object + ;`

Ex : `n2 = new Sample();`

3] Reference variable Declaration & Initilization in one line :

syntax : `classname reference variable = new constructor();`

Ex :

```
Sample n2 = new Sample()
n2.add();
n2.add();
n2.add();
```

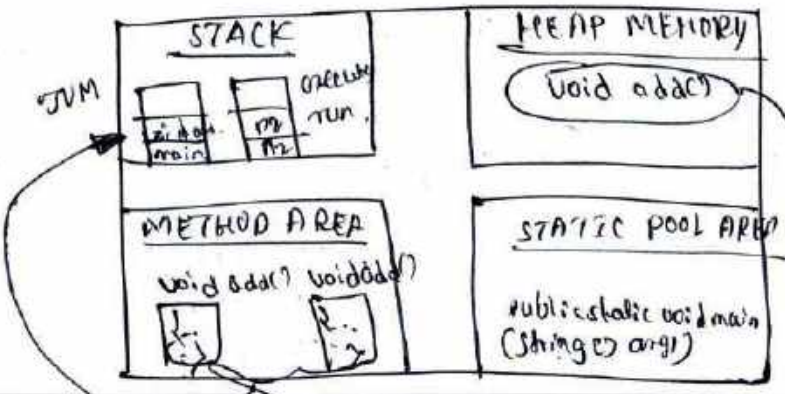
Example:

```
class Sample
{
    void add()
    {
        int a=10;
        int b=20;
        int c=a+b;
        System.out.println("The Addition of Two No's: "+c);
    }

    Public static void main(String[] args)
    {
        Sample n2 = new Sample();
        n2.add();
        n2.add();
    }
}
```

store res in reference variable

OUTPUT

```
The Addition of Two No's is : 30
The Addition of two No's is: 30
```



Write a Java program to access Data inside the class non-static with reference variable.

```
class Reference
{
  void square()
  {
    final double pi = 3.142;
    int r = 5;
    double res = pi * r * r;
    System.out.println("The Area of square is:" + res);
  }

  public static void main(String[] args)
  {
      Reference man = new Reference();
              man.square();
  }
}
```

OUTPUT

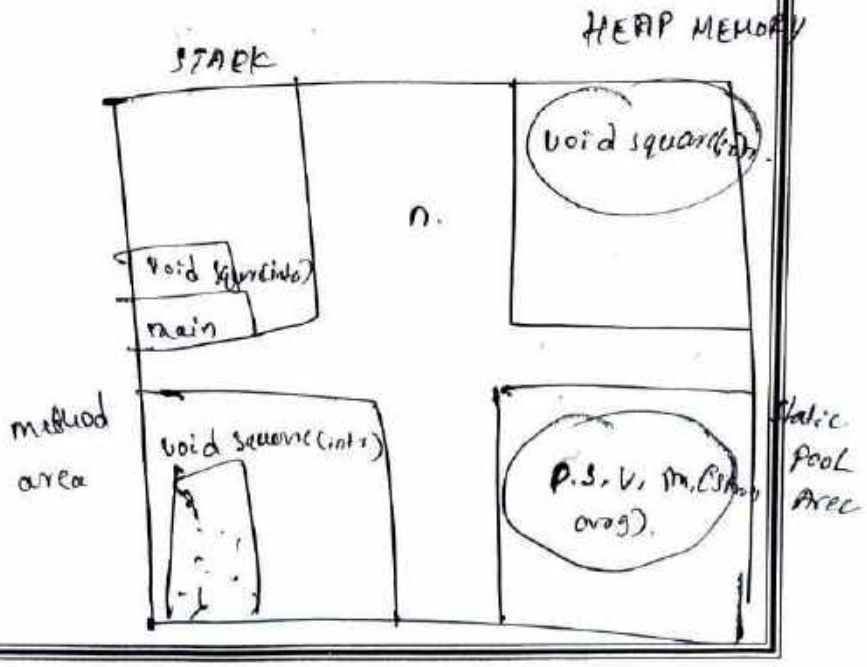
```
The Area of square is: 78.55
```

21/10/2022

Write a program from non-static to static through reference variable with parameters.

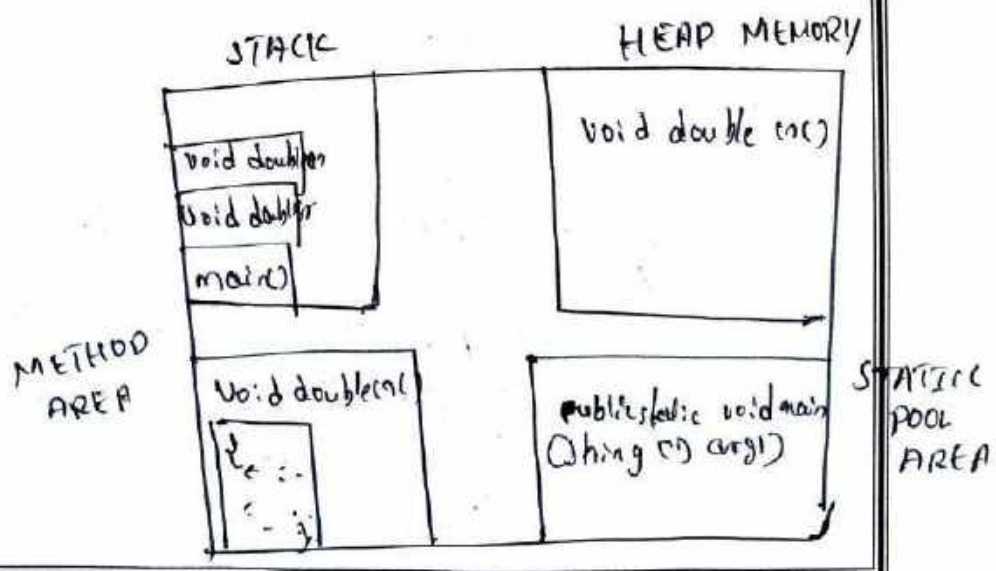
Area of square Pi*r*r

```
class Roler
{
    void square (int r);
    {
        final double Pi=3.142;
        double res = pi*r*r;
        System.out.println("The Area of square is:"+ res);
    }

    public static void main (String [] args)
    {
        Roler n = new Roler ();
        n.square (2);
    }
}
```



Write a program to calculate Area of Circle with method with Return type.

```
class Circle
{
    double cr()
    {
        final double pi = 3.142;
        int r = 2;
        double res = pi*r*r;
        return res;
    }

    public static void main (String[] args)
    {
        Circle m = new Circle();
        double x = m.cr();
        System.out.println(" The Area of circle is:" + x);
    }
}
```



Write a program to print reference variable and get the address of the object.

Whenever we print the reference variable it will be print the object address. That is folder name or package name . class name ~~ab~~ @ Hexa Decimal Numbers.

(K) It is also called as fully qualified path.

(K) we will write the object in a main method ~~class~~ but when ~~can't~~ ~~all~~ we declared the non-static of the members then only we need object.

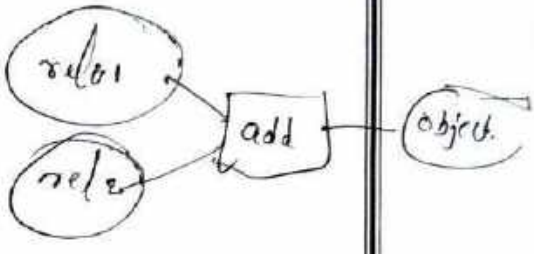
(K) otherwise ~~object~~ ~~a~~ we can't written a ~~obje~~ non static member of the class we write a object that's false but not executes the program and that gives the object address.

object 1 Address
object 2 Address

```
Class sample 1
{
    public static void main (String[] args)
    {
        sample s1 = new sample();
        sample s2 = new sample();
        S.O.P (s1) ----------> object1 Address
        S.O.P(s2) ----------> object2 Address
    }
}
```

2 objects store, we will get same object address.

```
class Sample1
{
```

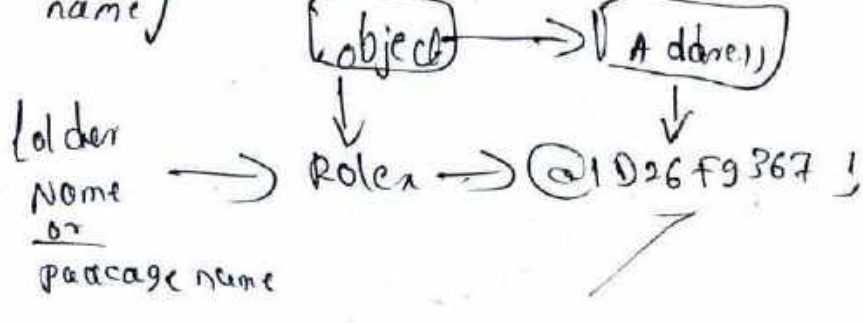



```
public static void main (String[] args)
{
    sample1 s1 = new sample1();
    Sample1 s2 = s1;
}
    System.out.println(s1);
    System.out.println(s2);
}
```

Object Address → full qualified path.

syntax:

folder name / package name } . class name @ hexa decimal Number.



SWITCH CASE :-

(1) Switch case is use for pattern matching in matches the particular direct the sequence and executes particular cases.

SYNTAX :-

```
Switch (char sequence)
{
    case1 :  statement 1
             break;
    case 2 :  statement 2
             break;
    case3 :  statement3
             break;
    Default :  statement;
             break;
}
```

```
class Rolex
{
  public static void main (String [] args)
  { ~~string~~ int a = 1;
    switch (a)
      case1 : System.out.println (" Hi ");
              break;
      case2 : System.out.println (" Hello ");
              break;
      default : System.out.~~println~~ (" Invalid ");
}  }
```

Sol (ch)

Enter a char (A)

```
class Sample
{
  public static void main (String[] args)
  {  string str = "Java";
     switch ( str )
     {
      "Java" : system.out.println(" Hi");
               break;
      "Mava" :  s.o.p ("Hello");
               break;
      default :  s.o.p ("Invalid");

     } }
}
```

22/10/2022

Looping statements:

for:

Write a program 1 to 10 by using for loop.

```
class Roler
{
    int i=1;

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

OUTPUT:

```
The Number is: 1
The Number is: 2
The Number is: 3
The Number is: 4
The Number is: 5
The Number is: 6
The Number is: 7
The Number is: 8
The Number is: 9
The Number is: 10
```

For loop

Decrement

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

OUTPUT

```
10
9
8
7
6
5
4
3
2
1
```

for loop multiplication :

```
for ( i=1 ; i<=10 ; i++)
{
    system.out.println(i*i);
}
```

i=1

1*1=1
2*2=4
3*3=9
.
.
n*n=n

Addition

```
for ( i=1 ; i<=10 ; i++)
{
    system.out.println(i+i)
}
```

```
for ( int i=1 ; i<=10 ; i++)
{
    system.out.println(i*i+i);
}
```

```
for ( int i=1 ; i<=10 ; i++)
{
    system.out.println(i*i-1)
}
```

22/10/2022

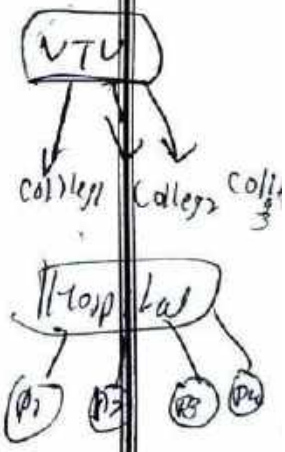
Real time examples on reference variable with non-static method :

① Sanju and geetha joined Jspiders in the first mock geetha got mock rating 1, Sanju got mock rating is 2. Sanju was frustrated late evening will meet his friend and his friend will give beautiful idea. study Hard whole night. and take remote. Sanju will get motivated he will study whole night and take remock. and he gets 1 as mock rating.

② Write a program to print how much sanju & geetha got the mock rating.

geetha mock rate = 1
Sanju mock rate = 2 after,

```
class Jspider
{
    int Java_mock;
static string in_name = "Javaspider";

public static void main (String [] args)
{
    Jspider name = new Jspider();
        name.Java_mock = 1;
    System.out.println (Java_mock);

    Jspider name = new Jspider();
        name.Java_mock = 2;
        name.Java_mock = 1;

System.out.println (Java_mock);
```



Java Spider
~~name~~ 1
1

```
class Education
{
    static String sc_name="saint joseph";
           String sc_branch;
           int sc_strength;


    public static void main(String [] args)
    {
        System.out.println("The school name is:"+ sc_name);
        Education n = new Education();
                n.sc_branch="rajajinagar";
                n.sc_strength=5000;
        System.out.println("The branch is:"+ n.sch_branch);
        System.out.println("The school strength is:"+n.sc_strength);
    }

}
```

OUTPUT:-

```
The school name is: saint joseph:
The branch is: rajajinagar.
The school strength is 5000
```

Operators

1] post fix operator $(a++) = (++a)$

2] pre fix operator $(a--) = (--a)$

$a = 10$

$a++ = 10$

$++a = 11$

int a=13;

S.O.P (a++ - ++a + ++a + a++ - ++a);

S.O.P (++a + ++a - a++ - a++);

S.O.P (a++ - ++a - a++ + a++ - ++a - a++);

S.O.P (++a - ++a - a++ - ++a + ++a + a++

+ ++a);

a=6

a++

++a

7

10

a++

++a

Unary operator :

1) post fix => a++ a-- / ++a --a

~~2) pre fix => ++a --a~~

1) post fix : is a unary operator is used to increment the values first utilizes the values after increment it.

for example. int a=7;

~~a++ + a++~~

~~7 8~~

a++ = 7+1 = 8

++a = 1+7 = 8

a++ + a++

7 + 8 = 15.

~~int~~ int a=5;

a++ + a++ + a++

5 + 6 + 7 = 18.

---

int a=5

int b= a++ + a++ + a++

5 6 7 = 18

a++ + a++ - ++a

5 + 6 - 7

11 - 7 = 4.

7+8

int x=5

int b= x++ + x++ + ++x

5 + 6 + 8 = 19

27/10/2022

CONSTRUCTOR :-

" Constructor is a special method, / special member of the class. which is used to initilize the data members."

RULES OF CONSTRUCTORS :-

1] Constructor name should be same as class name.

2] Constructor will not have any return type.

3] Constructor will not return any value.

4] Constructor is always Non-static.

5] Whenever an object is called constructor will get invoked.

CONSTRUCTOR SYNTAX :-

```
class class-name
{
    class-name() -
    {
        . . . . .
        . . . . .
    }
}
```

Constructor

Example:

```
class sample
{
    sample ()
    {
        system.out.println(" Hey I am constructor ");
    }

    public static void main (String[] args)
    {
        System.out.println(" *** --- MAIN STARTS --- *** ");
        new sample ();
        System.out.println(" *** --- MAIN ENDS --- *** ");
    }
}
```

constructor

OUTPUT

```
*** --- MAIN STARTS --- ***
Hey I am constructor.
*** --- MAIN ENDS --- ***
```

Write a program to initilize the variables through the constructor.

```
class Demo
{
    int x;
    Demo ( int y)
    {
        x=y;
    }

    public static void main (String [] args)
    {
        Demo d1= new Demo (10);
        System.out.println (d1.x);
    }
}
```

OUTPUT

10

```
class Demo1
{
    int x;
    String name;
    Demo1 (int y , String str )
    {
        x = y;
        name = str;

    }

    public static void main (String [] args)
    {
        Demo1 d2 = new Demo1 (20, "JAVA");
        System.out.println (d2.x);
        System.out.println (d2.name);

    }
}
```

OUTPUT

```
20
JAVA
```

```
class facebook
{
    string w_name;
    int pwd;
    facebook (string sh, int pwd)
    {
        w_name = sh;
        pwd = pwd;

    }

    public static void main (String [] args)
    {
        system.out.println ("---- MAIN STARTS ----");
        facebook fb = new facebook ("dinga", 1234);
        system.out.println (fb.uname);
        system.out.println (fb.pwd);
        system.out.println ("---- MAIN ENDS ----");
    }
}
```

OUTPUT

```
"..... MAIN STARTS ....."
        dinga
        1234
"..... MAIN ENDS ....."
```

28/10/2022

Write a program to initilize Id and student name through contructor.

```
class student
{
    int std_id;
    String std_name;
    student (int s, String n)
    {
        std_id = s;
        std_name = n;
    }

    public static void main (String [] args)
    {
        student s1 = new student (11, "NAGU");
        System.out.println (s1.std_id);
        System.out.println (s1.std_name);
    }
}
```

OUTPUT

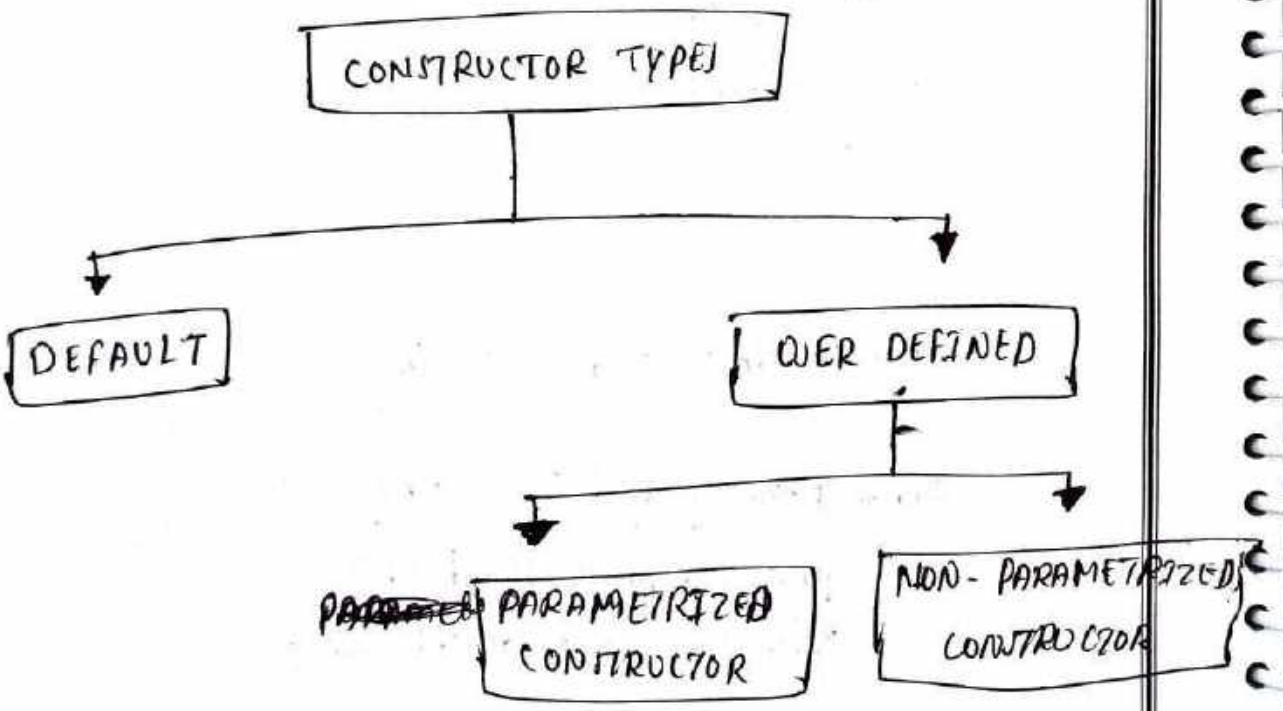
```
11
NAGU
```

: CONSTRUCTOR TYPES :-

1] Default

2] User-defined.

- parametrized constructor.
- Non-parametrized constructor.



Write a program to initilize school name, school strength and school fees using constructor.

```
class school
{
  string s_name;
  int s_strength;
  int s_fees;
  school (string x, int y, int z)
  {
     s_name = x;
     s_strength = y;
     s_fees = z;

  }

public static void main (string[] args)
{
   school r = new school ("SSLC", 5000, 20000);
     system.out.println (r.s_name);
     system.out.println (r.s_strength);
     system.out.println (r.s_fees);
}

}
```

OUTPUT

```
SSLC
5000
20000
```

THIS: This is a keyword & is used in constructor. when even we initilized same values, we confirmation for this is the ~~new~~ new value.

ex: stud_id = stud_id;

same value

we confirm the which one is original then why we use this keyword.

```
this. stud_id = stud_id;
```

for example:

```
class Employee
{
  string emp_n;
  int emp_id;
  Employee (String emp_n, int emp_id)
  {
    this. emp_n = emp_n;
    this. emp_id = emp_id;

  }

  public static void main (String[] args)
  {
    Employee E = new Employee ("NAGU", 11);
    System.out.println (E.emp_n);
    System.out.println (E.emp_id);
  }
}
```

OUTPUT

NAGU
11

This. Keyword :

- It is used whenever the local and global variables names same. to differentiate b/w them we use this keyword.
- This. keyword is also called as default reference variables.
- This keyword can be used only the non-static contexts because this keyword itself a non-static.
- This keyword is used to point to current object.

30/10/2022

for looping stmt: examples:

Tot cases

```
class for
{
    int i;

    public static void main (String [] args)
    {
        for (i=1; i<=5; i++)
        {
            system.out.println(i*i+5);
                                1*1+5=6
                                2*2+5=9
        }
    }
}
```

o/p

6
9
14
21
30

```
for( i=1; i<=5; i++)
{
    system.out.println(i*i*i+2);
                        /* 1*1*1+2 =3
                           2*2*2+2= 10
                           3*3*3+2 = 29 */
}
```

1*1*1 =
2*2*2 =
3*3*3 =

1*1*1+1 = 1
2*2*2+1 = 9
1³ = 1*1*1+2=3
2*2*2+2=10
3*3*3+2=29

```
for (i=1; i<=5; i++)
{
    system.out.println(i*4);
                        1*4 = 4
                        2*4 = 8
                        3*4 = 12
}
```

4
8
12
16
20
24

1*4
2*4
3*4

1*1*1*1+3 = 4
2*2*2*2+3 =

30/10/2022

COMPOSISTION :-

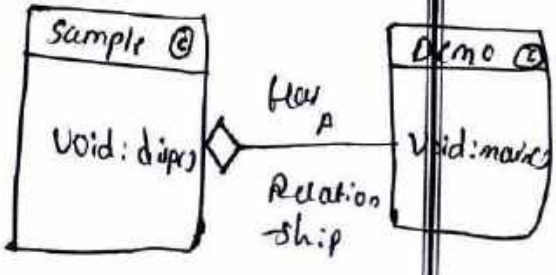
- Composition means, A class having object of another class is called as composition.
- It is also called Has a realtionship.
- class Diagram is a pictorial representation. to represent the member of the class.

```
class Sample
{
    void disp()
    {
        System.out.println("Hello composition");
    }
}
class Demo
{
    public static void main (String [] args)
    {
        Sample s1 = new Sample();
        s1.disp();
    }
}
```

O/P

Hello composition

Class Diagram



Pass by value And pass reference :-

PASS BY VALUE :-

(*) calling / invoking method by passing premitive type of datas is called as pass by value. / call by value.

```
class sample
{
  static void ~~disp~~add (int a)
  {
    System.out.println (a+a);
  }

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

10

10+10 = 20

Primitive And Non primitive type of pgm

1/11/2022

Primitives

1]
```
class Demo
{ ~~int a=10;~~
  psvm(__)
  { int a=20; //local variable.
  }
}
```

2]
```
class Demo
{ int a=20; //global variable
  psvm(__)
  {
  }
}
```

Non-primitives

1]
```
class Demo
{
  Demo d1=new Demo();
  psvm(__)
  { Demo d2=new Demo();
  }
}
```

2]
```
class Demo1
{
  Demo1 d1=new Demo1()
  psvm(__)
  { Demo1 d2= new Demo1();
  }
}
```

primitives methods & non primitive method

```
class sample
{ void area()
  r=0.5
  s.o.p(area);
  psvm(__)
  { area();
  }
}
```

→

```
class sample
{
  ~~static~~ void add()
  { sample t1=new sample()
    s.o.p(t1);
  }
  psvm(__)
  {
    add();
  }
}
```

primitives method with parameters & non primitive method with parameters

```
class sample
{
  static void add(int a, int b)
  c=a+b
  s.o.p(c);
  p.s.v.m(__)
  {
    add(10,10);
  }
}
```

```
class sample
{
  static void add(sample s)

  p.s.v.m(__)
  { sample s1=new sample()
    add(s1)
  }
```

mimitives method with return type and non-mimitives method with return type

```
class Sample3
{
static double area()
{double res = 89.56;
 return res;
P.S.V.m(——)
{
 double x = area();
 S.O.P(x)
}
}
```

```
class Sample4
{
static sample4 ~~double~~ area()
{
 sample4 res = new sample4
 return res
P.S.V.m(——)
{
 sample4 x = area();
 S.O.P(x);
}
}
```

---

reference by value :- calling a reference value is called as reference by value.

PASS BY REFERENCES :-

① calling or invoking the method by passing the reference variable is called as ~~reference by c~~ Pass by reference.

② pass by references is indirectly pass by value.

primitive pass by value	non-primitive pass by reference.

Class Demo
{
static void add (int a)
{
sop(a+a);
}
P S V m (S[] a)
{
int x=10
add (x);
}
}
	class Sample
	{
	int a=30;
	static void dingi(sample s2)
	{
	sop(s2.a);
	}
	P.S.V.m(S[] a)
	{
	Sample s1 = new sample()
	sop(s1.a)
	dingi(s1);
	}

Real time example on pass by reference:-

1) In IBM company HR, and Employee is there. the IBM HR is opening for job application. The IBM HR ask to employee Hiring some candidates. The candidates are looking for Jobs. They will apply the Jobs.

(2) pass by reference is a calling method / invoking by passing reference variable is called as pass by reference.

(3) referring the one person to another. ....

```
class Aunty
{
    void job ()
    {
        system.out.println("Java Development");
    }
}
class Uncle
{
    public static void main (String[] args)
    {
        Aunty a1 = new Aunty();
        Ding.needjob(a1);
        Ding1.needjob(a1)
    }
}
class Ding
{
    static void needjob (Aunty a2)
    {
        a2.job();
    }
}
class Ding1
{
    static void needjob (Aunty a3)
    {
        a3.job();
    }
}
```

4/11/2022

CLASS AND OBJECT

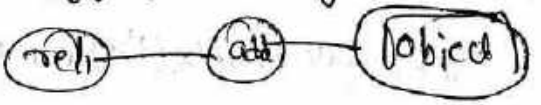
- **Class:** class is a blue print or template to create object.

- **Object:** object is a realtime entity, which has its own state and behaviour.

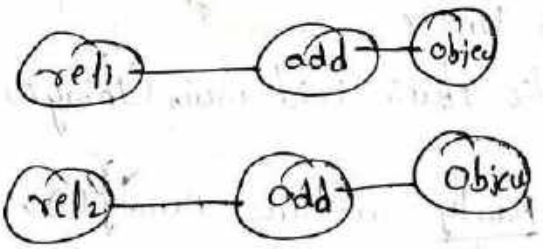
- Non-static variables are the state of the object.
- Non-static methods are the behaviour of the object.

- Whenever, we can create object it will have both state and behaviour.

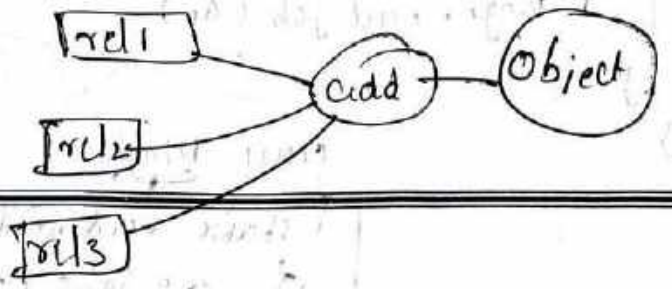
- state defines what data it can hold.
- behaviour defines, the way it can behave.
- all the object address will get stored in reference variable.



- Multiple objects will get stored in multiple reference variable.



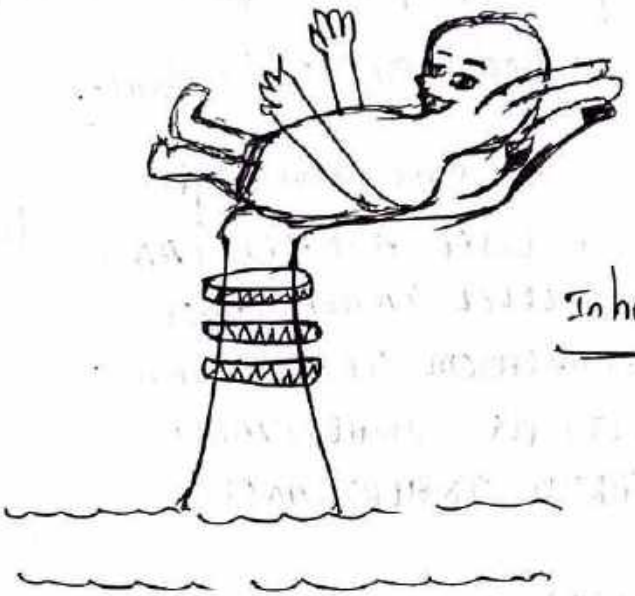
- object address can be stored in multiple reference variable.



INHERITANCE

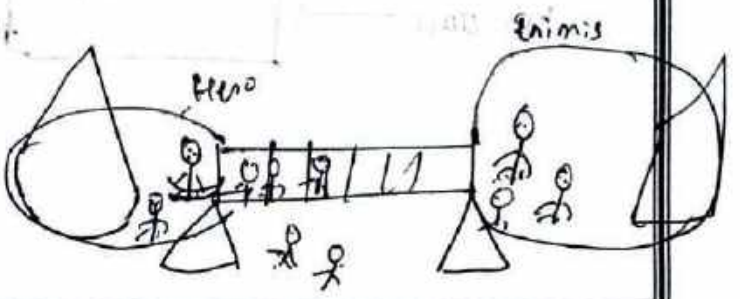
4/11/2022

(R) Inheritance means it properties of next version.

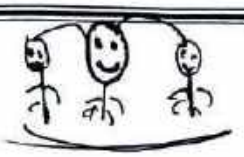


Inheritance

Inheritance



Multiple Inheritance



Inheritance

Krish K1, K2, K3

Inheritance :

(1) "Inheriting the properties from one class to another class is called as Inheritance."

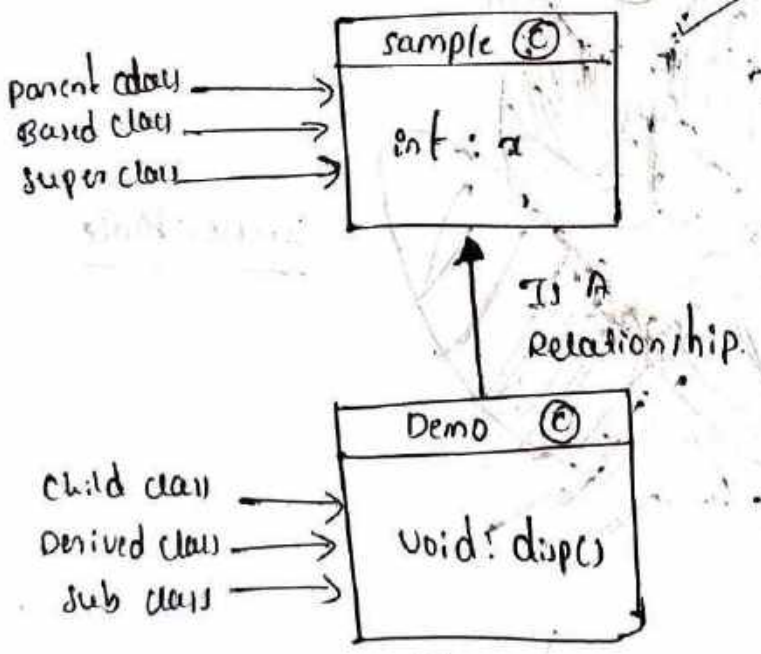
(2) Inheritance we have five types

1) SINGLE LEVEL INHERITANCE
2) MULTI LEVEL INHERITANCE
3) HIERARCHICAL INHERITANCE
4) MULTIPLE INHERITANCE.
5) HYBRID INHERITANCE.

" when do we go for Inheritance. code reusability

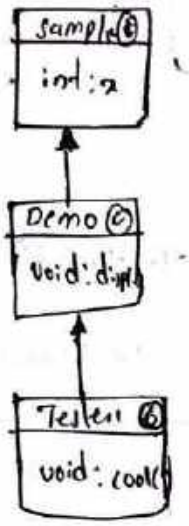
1) SINGLE LEVEL INHERITANCE :

(1) A sub class inheriting the properties from only one super class. is called as single level inheritance.



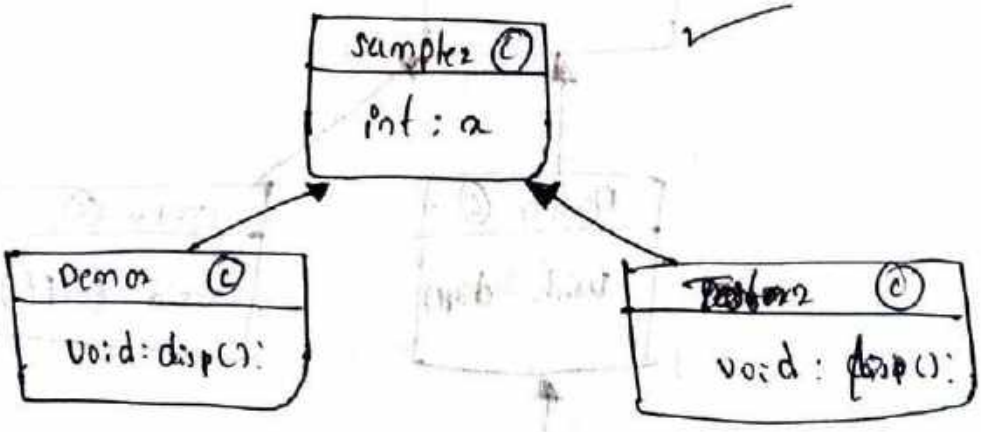
2) MULTI LEVEL INHERITANCE :-

A sub class Inheriting the properties from it super class. Intern super class inheriting the properties from its super class. is called as multi level Inheritance.



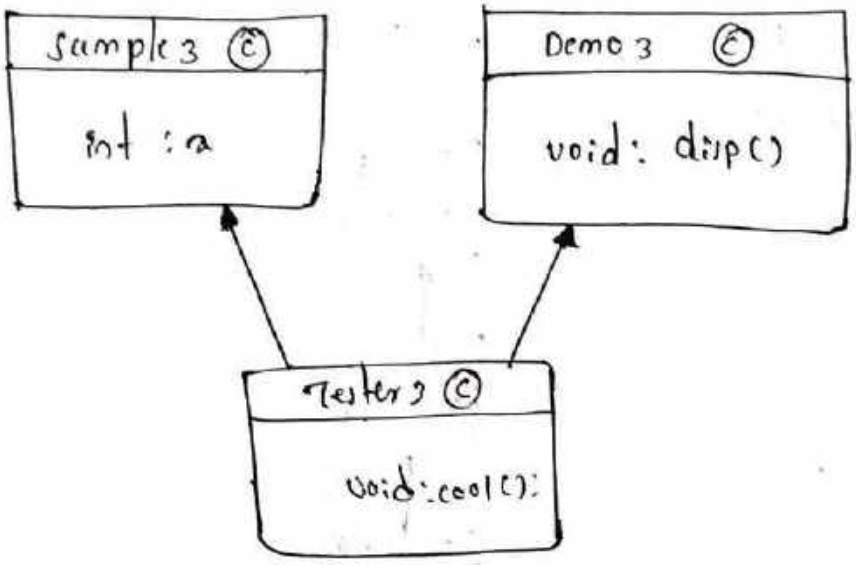
3) HIRARACHICAL INHERITANCE :-

multiple sub class inheriting the properties from only one super class. is called Hierarchical inheritance.



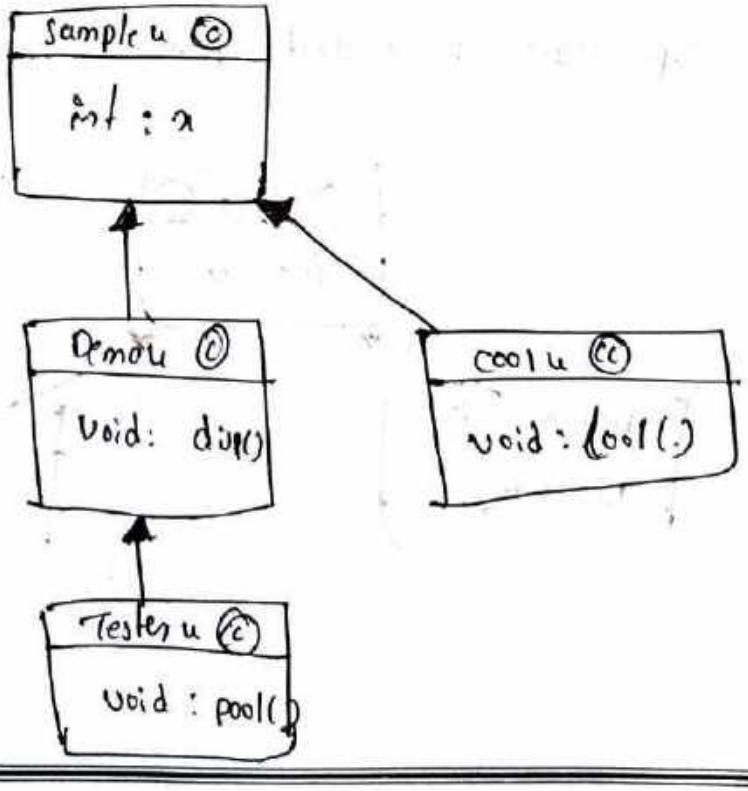
4) MULTIPLE INHERITANCE :

A sub class inheriting properties from multiple super class is called as multiple Inheritance.



5) HYBRID INHERITANCE :

It is a combination of single level, multi level and Hierarchical Inheritance is called as Hybrid Inheritance.

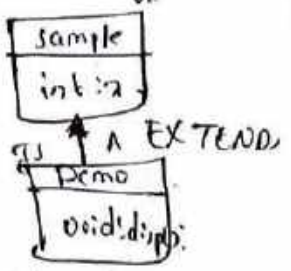


NOTE

① Multiple Inheritance is not possible in Java through class. But we can Achive through Interface.

② we go for Inheritance "code reuseability"

1) Single level inheritance :- sub class inheriting the properties from only super class.



ex:

```
class sample
{
    int x=10;
}

class Demo extends sample
{   // int x=10;
    void disp()
    {
        System.Out.Println("Hello");
    }
}

class main class
{
    public static void main (String[] args)
    {
        Demo d=new Demo();
        d.x;        int x=10;
        d.disp();   Hello
    }
}
```

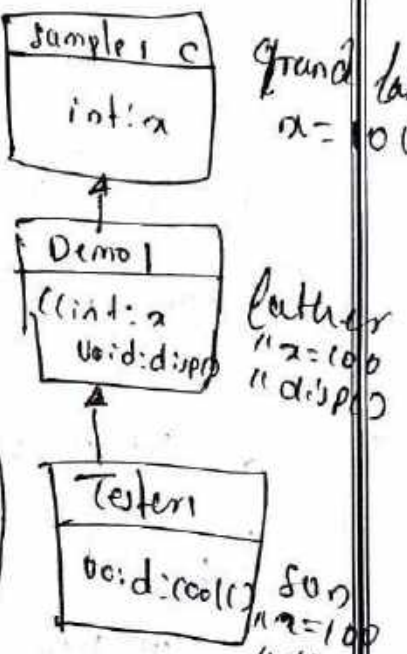
① ② ③

2) Multi level Inheritance : multi level inheritance.

A sub class Inheriting properties from superclass. Intern a super class Inheriting properties from main super class is called as multi level Inheritance.

ex:

```
class sample
{
    int : a
}

class Demo1 extends sample
{
    void disp()
    system.out.println("it's display");
}

class Tester1 extends Demo1
{
    void cool()
    {
        system.out.println("Hey cool");
    }

class Main class 2
{
    public static void main (String [], args)
    {
        Tester1 t=new Tester1 ();
        t.a;
        t.disp();
        t.cool();
    }
}
```

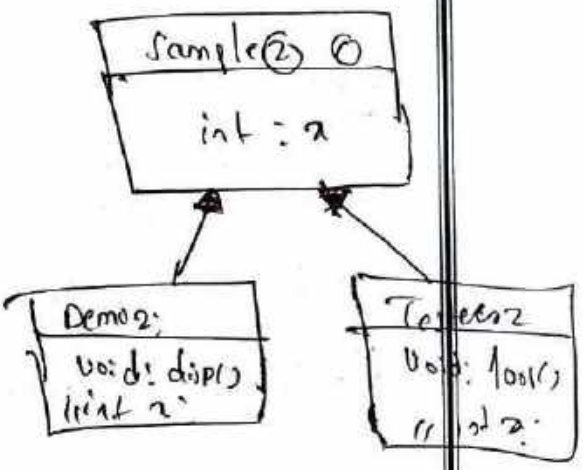


(2)

(1)

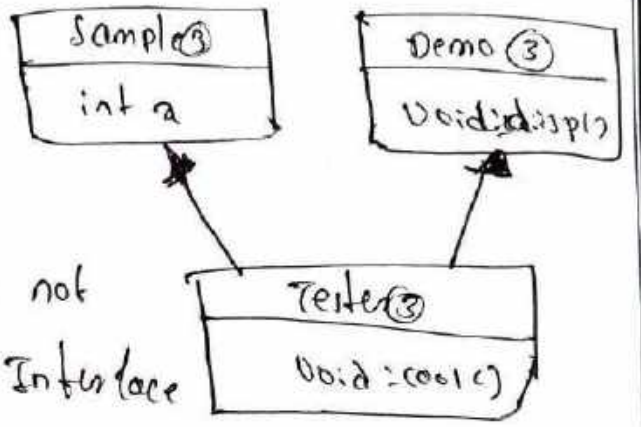
3] Hierarchical Inheritance :- Hierarchical inheritance is ~~a sub class~~ multiple sub class inheriting the properties from only one super class / common class is called Hierarchical Inheritance.

ex :




```
class sample
{
  int x=10;

}
class Demo2 extends sample
{
  void disp()
  {
    S.O.P("Hi");
  }
}
class Tester2 extends sample
{
  void foo()
  {
    S.O.P("Hello");
  }
}
class Main class3
{
  public static void main (String [] args)
  {
    Demo2 D= new Demo2();
          D.x;
          D.disp();
    Tester2 t=new Tester2();
          t.x;
          t.foo();
  }
}
```

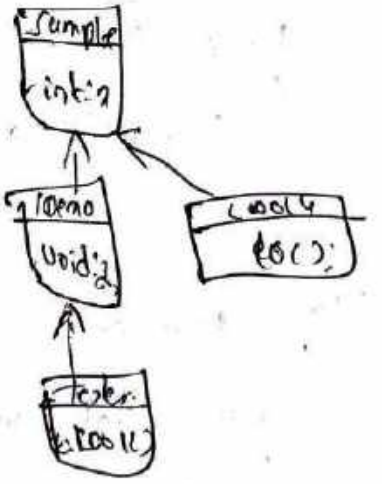
4) Multiple Inheritance: ~~As super~~ A Sub Class Inherting properties from multiple super class is called as multiple Inheritance.



Note:

multiple Inheritance is not used class, its we Interface through executes its.

5) Hybrid Inheritance: Hybrid inheritance is a combination of single level, and multi level, and Hierarachical ~~level~~ Inheritance is called Hybrid Inheritance.



7/11/2022

BLOCKS :

- Blocks are used to initilize the variables.
- In blocks we have two types
  1] SIB (Static Initilization Block).
  2] IIB (Instance Initilizatio Block).

1] SIB (Static Initilization Block) :-

- Any block which declared in a class with keyword static is called as static Initilization Block.
- SIB will get executed before main method.
- we can have 'n' number of SIB, the order of execution is sequential.

~~2] IIB (Inst~~

SYNTAX :-

```
class classname
{
    static {
        stmt 1 ;
    }

    public static void main (String [] args)
    {
        S.O.P("stmt 1");
        system.out.println (--- Main method ...)
        system.out.println(---- Main ends ...)
    }
}
```

stmt1
Main method
Main ends

Ex:

```
class Demo
{
    static int a;
    static {
        int a = 100;
    }

    public static void main(String[] args)
    {
        system.out.mintln (a);
        System.Out.println(" Main starts");
        system.out.println("Main ends");
    }
}
```

O/P

100

2) IIB

(R) Any Block which Declared in a class without using keyword static, is called as Instance Initilization Block.

(R) IIB will get executed whenever we can object created.

(R) we can have 'n' number of IIB's, the order of execution is sequential.

Ex:

```
class sample
{
    int a;

    {
        a = 100;   //non static variable
                   initilization.
    }

    public static void main (String[] args)
    {
        system.out.println ("Main starts");
        new sample ();
        system.out.println ("Main ends");
    }
}
```

Main starts
100
Main End,

ORDER OF EXECUTION :

```
class Demo
{
    Demo()
    {
        system.out.println("Hey I am constructor");
    }

    static {
        system.out.println("Hey I am static Block");
    }

    {
        system.out.println("Hey I am non static Block")
    }

    public static void main (String [] args)
    {
        S.O.P ("Main starts");
        new Demo();
        S.O.P ("Main Ends");
    }
}
```

O/P

```
Hey I am static Block
Main starts
Hey I am non-static Block
Hey I am constructor
Main Ends
```

9/11/2022

METHOD OVERLOADING:

"Developing multiple methods with the same name, but variation in the Arugument list is called as Method overloading."

Variation in the Argument list means below

- variation in the data types
- variation in the length of the Arguments
- variation in order of occurance in the Arguments

When to go for method overloading

Whenever we want to perform common task or operation then we go for method overloading.

we can overload non-static method and static and main method.

Rules:

- The method name should be same.
- Their should be argument list ~~in variation~~.
- Their is no ~~restrion~~ restriction on access specifier, modifier and return type.

Assignment : Real time ex. on method overloading

Note:

Example :-

```
class Whatsapp
{
    void send (int no)
    {
        ~~S.O.P~~ system.out.println ("sending no" + no);
    }

    void send (String txt msg)
    {
        system.out.println ("sending txt msg" + txt msg);
    }

    void send (int no, String txt msg)
    {
        system.out.println ("sending no with txt msg" + no, txt msg);
    }

    void send (String txt msg, int no)
    {
        system.out.println ("sending ~~no with~~ txt msg with no" + txt msg, no);
    }
    public static void main (String[] args)
    {
        Whatsapp w1 = Whatsapp ();
        w1.send (11);
        w1.send ("Hi");
        w1.send (111, "Hello");
        w1.send ("NICE", 120);
    }
}
```

O/P

```
11
Hi
111 Hello
NICE 120
```

```
class phonepay
{
  static void pay (int no)
  {
   system.out. println (" pay by contact Number");
  }

  static void pay ( string c_name)
  {
   system.out. println (" pay by contact Names");
  }
  static void pay ( int no, string c_name)
  {
   system.out. println (" pay by Number with contact Name");
  }
  static void pay (string c name, int n)
  {
   system.out.println (" pay by ~~Number with~~ contact name with Number");
  }
}

class Main class
{
  public static void main (String [] args)
  {
      phonepay. pay (11);
      phonepay. pay ("NAGARAJA");
      phone pay. pay (11, "NAGARAJA");
      phone pay. pay ("NAGARAJA", 11);
```

O/P

```
Pay by 11
Pay by NAGARAJA
Pay by 11, NAGARAJA
~~Pay by NAGARAJ~~ 11
```

```
  }
}
```

Main method over loading :

```
class MainClass
{
    public static void main (int a)
    {
        System.out.println (" Number a is "+ a);
    }

    public static void main (String text msg)
    {
        System.out.println (" ~~Number~~ text is " + text msg);
    }

    public static void main (int a, String text msg)
    {
        System.out.println" Number with text msg "+ a, text msg);
    }

    public static void main (String text msg, int a)
    {
        System.out.println (" text msg with Number "+ text msg, a);
    }

    public static void main (String [] args)
    main (10)
    main ("Hello");
    main (11, "Hello");
    main ("Hello", 11);
}
```

9/4/2022

ARRAY :-

* Array is Linear Data structure which is used to store Homogenious type of data.

Drawbacks of Array:-

* size is fixed.
* It's can store only homogniow type of Data.

Array homogineous type of data:

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

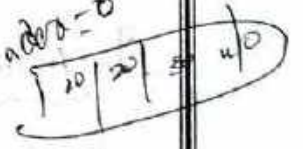
→ homogineous type of Data
homogineous means same type of data.

* Array is used to store the values.
* Array is simple and easy to use.

1] Array Declaration :-

syntax :- datatype [ ] array name ; // array

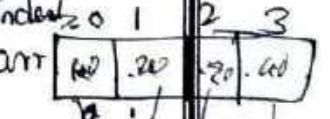
ex :- int [] arr ;



2] Array Initilization :-

syntax :- array name = new datatype [size] ;

ex :- arr = new int [4] ;

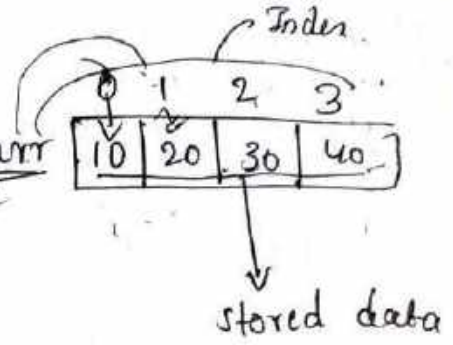


3] store data in array :

Syntax :- array name [Index] = values ;

ex :

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



4] Array Declaration and Initilization in one line :-

Syntax : data type [] array name = new datatype [size];

ex : int[] arr = new int[4];

5] Array utilization :- // to print array data give the array Index address.

```
System.out.println (arr[0]); 10
system.out.println (arr[1]); 20
// system.out.println (arr[2]); 30
System.out.println (arr[3]); 40
```

6] Array optimization :-

for loop :

```
int[] arr = new int[4]; // array Declaration initilization
for (int i=0; i<=4; i++)
{
    System.out.println (arr[i]);
}
```

arr [0] = 10 — 10
arr [1] = 20 — 20
arr [2] = 30 — 30
arr [3] = 40 — 40

7) To find the Array size :-

(*) Array size is fixed, we can't change it when we declared,

(*) we can find array size through the name is length — arr.length

Syntax :- array name . length ;

ex :- System.out.println (arr.length);

8) Array ~~value~~ Re-Initilization :- updated value.

Syntax : array name [Index] = new value ;

ex : arr [0] = 100 ;

Array without New operator :- Direct Values Assigning

Syntax : data type [] arr = { value1 , value2 , value3 . . value n};

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

Copying of one Array Data to store another Array.

int[] akk = {10, 20, 30, 40}; akk [10 | 20 | 30 | 40]

int[] abb = akk; abb = akk

abb [10 | 20 | 30 | 40]

Example on Array :-

①

```
class Array
{
    public static void main(String[] args)
    {
        int[] arr = new int[4];
        arr[0] = 10;
        //arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;

        for (int i=0; i<=4; i++)
        {
            system.out.println(arr[i]);
        }
    }
}
```

(2)

```
class Array1
{
    public static void main (String[] args)
    {
        int[] arr = {10, 20, 30, 40, 50};
        for (int i=0; i<=5; i++)
        {
            system.out.println(arr[i]);
        }
    }
}
```

14/11/2022

~~METHOD OVERLOADING:-~~

METHOD OVERRIDING :-

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

Method overriding rules :-

- The method name & signature should be same in the sub class as in the super class.
- The method should be non-static.
- Their should is a relationship inheritance.

When do we go for method overriding :-

- Whenever, we want to provide new implementation for existing feature. Then we should go for method overriding.

Note: existing software we can upgrade the software, is called as method overriding.

App Name: Whatsapp-V1
Status: test.
Whatsapp -V2 text, audio, video

for example :

```
class whatsapp_v1
{
    void status ( )
    {
        system.out.println (" status with text ");
    }
}

class whatsapp_v2 extends whatsapp_v1
{
    void status ( )
    {
        system.out.println (" status, text, images, videos ");
    }
}

class Main class
{
    public static void main (String[] args)
    {
        whatsapp_v2 v2 = new whatsapp_v2 ();
        v2.status();
    }
}
```

O/P

Status, text, images, videos

METHOD OVER HIDINING :-

super keyword

(i) In the case of method overriding Along with sub class implementation if we need super class implementation then we should use super. method name.

(ii) Method Hiding :

(i) In the case of method overriding the super class implementation is in hidden with the sub class implementation. and it is called as method Hiding.

Example :-

```
class phonepay_v1
{
    void pay()
    {System.out.println("pay with contact");
    }
}

class phonepay_v2 extends phonepay_v1
{
    void pay()
    {system.out.println("pay with scan");
     super.pay();
    }
}

class mainclass
{
    public static void main(String[] args)
    {
        phonepay_v2 v2 = new phonepay_v2();
        v2.pay();
    }
}
```

NOTE :

what is difference b/w this. keyword and super keyword

THIS KEYWORD	SUPER KEY WORD
⊛ this. keyword is used point the current object.	⊛ super keyword is used in case of method over riding.
⊛ whenever the local and global variable names are same. to differentiate b/w the we use this. keyword.	⊛ super keyword in the case of method over riding Along with the sub class implementation. if we need super class implementation then we go for super keyword.
⊛ this key can be used only in non-static contacts	⊛ super keyword can be used only in the case of method overriding.

TYPE CASTING :-

□ → O → P – A

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

* In Type casting we have two types

1) PRIMITIVE TYPE CASTING
   - Widening → implicity
   - Narrowing → explicity → explicity

2) CLASS TYPE CASTING / DERIVED TYPE OF CASTING.
   - up casting → implicity. → explicity.
   - Down casting → explicity.

1) PRIMITIVE TYPE CASTING :-

* converting from one primitive datatype to another primitive type is called as primitive type casting.

* Widening :-

converting from smaller ~~primitive data type~~ to any of its ~~attects~~ bigger primitive data types is called as widening

* widining can be done both implicitly and explicitly.

Narrowing :-

* converting from bigger primitive data type to any of its smaller primitive data type is called as Narrowing.

* Narrowing should be always done explicitly.

Example for widining :-

* implicitly :- it is internally executes code that implicitly.
* Explicitly :- it is executes the code over and visible stmts is called as explicitly

* implicitly

1) double x = 10;
SOP(x) o/p
10.0

2) float y = 100;
SOP(y) o/p
100.0f

3) double y = 31.3f
SOP(y) o/p 31.30

* Explicitly

1) double x (double) = 10;
SOP(x) : o/p
10.0d.

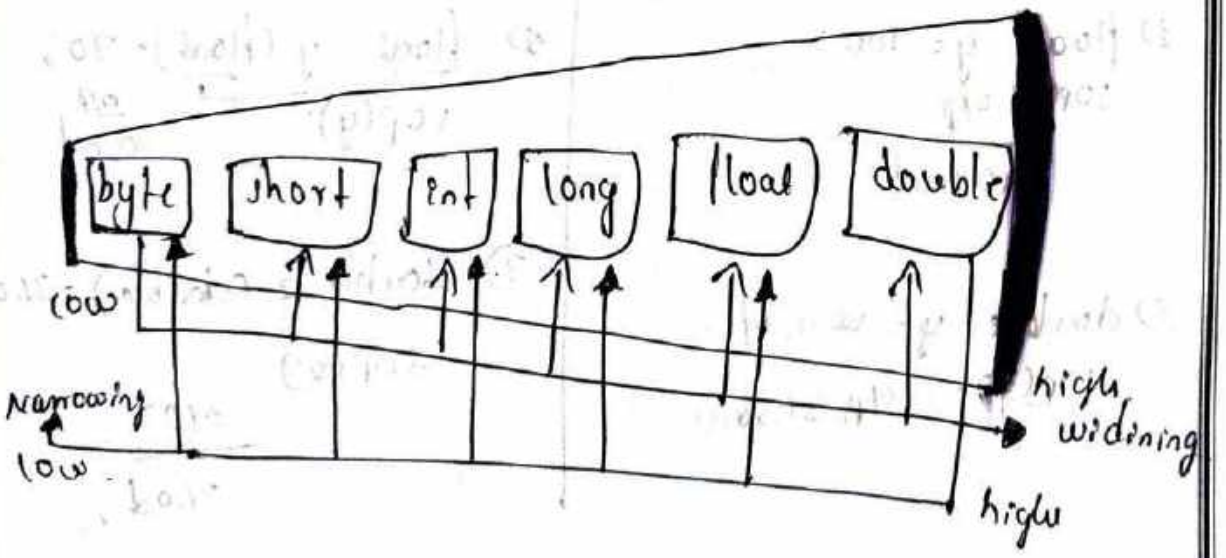
2) float y (float) = 30;
SOP(y) : o/p
30.0f

3) double z (double) = 31.0
SOP(z)
o/p
31.0d

Examples of narrowing:

Explicitly: over sides executes only.

```
int x = (int) 59.9d;
SOP(x);      O/P 59 //

byte a = (byte) 36.6f
SOP(a) 36 //

short y = (short) 37.39d
SOP(y) = 37 //

long z = (long) 91.37f;
SOP(z);
91 //
```



3] Class Type casting :- [Derived Type casting :

(*) Converting from one class object to another class type is called as class type casting.

1] UPCASTING :- "converting from sub class object to super class type is called as upcasting."

(*) Once, upcasted we will get only super class properties.

(*) upcasting ~~both~~ can be done both implicitly and explicitly.

```
Sample S1 = new Demo();
```

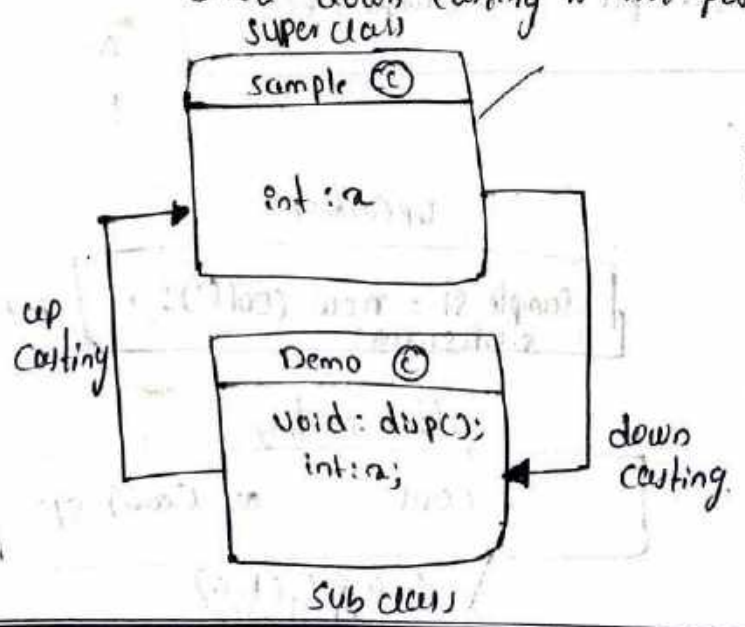
2] DOWNCASTING :- "converting from super class object to sub class type is called down casting."

(*) down casting ~~can~~ should be done explicitly.

(*) without upcasting we can't be ~~done~~ down casting.-

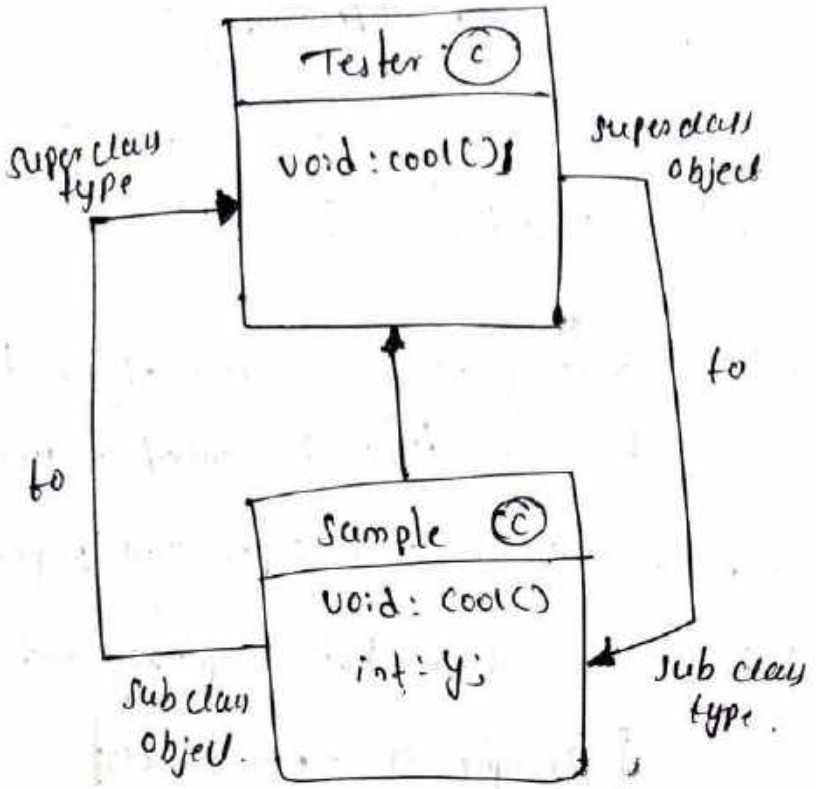
Direct down casting is not possible.

```
Demo d1 = (Demo) S1;
```



Samp class
(*) 1 properties
int a;

Demo class
(*) 2 properties
void : disp();
int a;



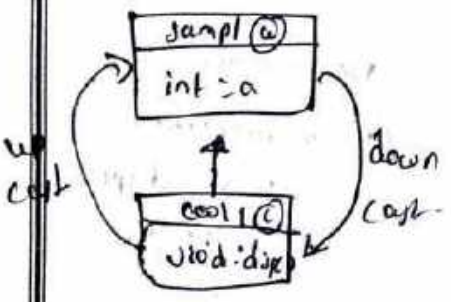
class Main class
P-1. v. m(S_a)

up casting.

```
Tester t1 = new Sample();
t1.cool();
```

down casting

```
Sample s1 = (Sample) t1;
```



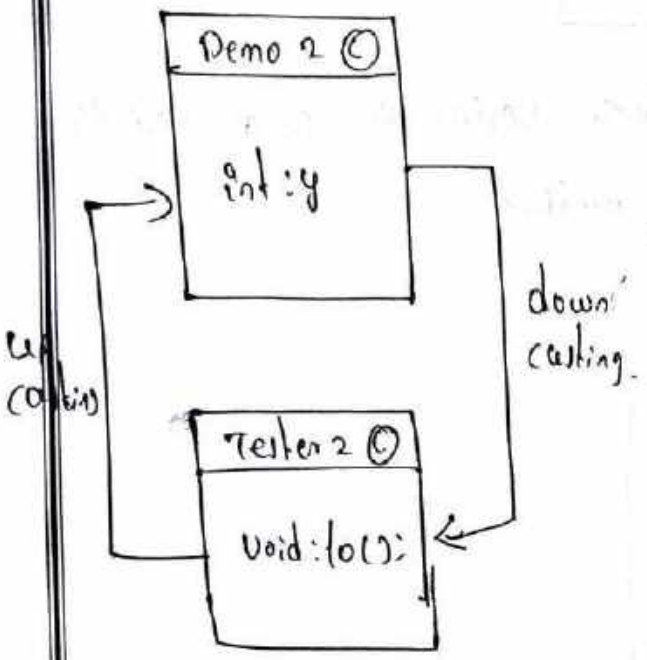
up casting

```
Sample s1 = new cool();
s.o.p(s1.a);
```

int a;

down casting

```
cool c1 = (cool) s1;
s.o.p(c1.a)
c1.disp();
```



up casting

```
Demo2 D2 = new Tester();
S.O.P(D2.y);
```

down casting

```
Tester t1 = (Tester) D2;
S.O.P(t1.y);
t1.fo();
```

Assignment:

Write 14 programs to do upcasting & down casting.

ABSTRACT

1) CONCRETE METHOD :-

(*) Any method which has both Declaration and Definition is called as Concreate method.

Concrete method :-

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

2) CONCRETE CLASS :-

(*) Any method | concrete method which is declared inside the class. is called as Concrete class.

Concrete class :-

```
class sample
{
    void test()
    {
      ____
      ____
      ____
    }
}
```

(*) Any class which has Concreate method is called concreate class.

3] ABSTRACT METHOD

(*) Any method which is declared with keyword abstract is called as Abstract method.

(*) it's doesn't contains body.

```
Abstarcd void test ();
```

abstract void add();

4] ABSTRACT CLASS :-

(*) Any class which is declared with the key word abstract is called as abstract class.

```
abstract class Demo
{

}
```

5]

(*) if class consist of abstract method, then the class should be ~~abstract~~ declared as abstract. is called as both class & method abstract.

(*) But vice-versa is not true.

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

6)

(*) An abstract class can have combination of concreate method and abstract method. or it have can only abstract class.

```
abstract class sample
{
abstract void cool()
        void disp()
        {
            s.o.p ("Hi");
        }
}
```

7)

(*) we can't declared abstract method as static, final, and private.

(*) The class which provides implementation for the abstract method. is called as implementation class.

NOTE:

(*) we can't create object for abstract class.

(*) if any one of the abstract method is not overriding in a sub class then the sub class should declared as abstract.

Program :

```
abstract class Sample2
{
abstract void disp();
abstract void test();
}

class Demo2 extends Sample2
{
    void disp()
    {
        S.O.P("Hi");
    }
    void test()
    {
        S.O.P("Hello");
    }
}

class Main class
{
    public static void main(String[] args)
    {
        Demo2 d2 = new Demo2();
        d2.disp();
        d2.test();
    }
}
```

O/P

Hi
Hello

```
abstract class Tester3 —
{
abstract void disp();
abstract void cool();
}

abstract class Demo3 extends Demo3 Tester3.
{   // abstract void disp();
    void cool()
    {
    s.o.p ("I am cool");
    }
}

class. Samples extends Demo 3.
{
    void disp()
    {
    s.o.p("Hi");
    }
}

Public static void main (String [] args)
{

    Sample3  s3 = new Sample3();
             s3.disp();
             s3.cool();

}
```

```
Hi
I am cool
```

INTERFACE :

(1) INTERFACE is a one of JAVA type.

(2) Interface have pure abstract body

[JAVA TYPE :] → class, enum, interface

In Interface we have two members :

1) Variables 2) methods

(3) All the variables in a interface by default it declared public static and final.

(4) All the methods are by default public abstract.

SYNTAX OF INTEFACE :

```
interface interface name
{
    variable;
    method();
}
```

Ex :

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

by default

by default

C → C extends

I → C implements

I → I extends

(5) In Interface we can't develop any concrete method

(6) Interface doesn't support constructor.

(7) Java provides keyword called implements to inherite the properties from Interface to class.

(1) The class which provides the implementation for the abstract methods is called as implementation class.

(2) If any one of the abstract method not overriden then the sub class should be declared as abstract.

(3) we can't create object for ~~Interface~~ abstract class and Interface. but we can create reference variable.

(4) To inherite interface to interface we used keyword extends.

(5) each and every class ~~except~~ extends object class. object class super most class in class type.

(6) Interface will never ever inherit the properties from a class, interface itself is the supermore

(7) Through interface we can achive 100% abstraction.

(8) Through abstract class we can achive upto ~~%~~ 100% abstraction.

Program :-

```
abstract interface sample1
{
    public abstract void disp();
    public abstract void test()
}

class Demo1 implements sample1
{
    public void disp()
    {
        s.o.p ("Hi");
    }
    public void test()
    {
        s.o.p ("Hello");
    }
}

class Main class
{
    public static void main (String [] args)
    {
        Demo1 d1 = new Demo1 ();
        d1. disp();
        d1. test();
    }
}
```

O/P

```
Hi
Hello
```

2)

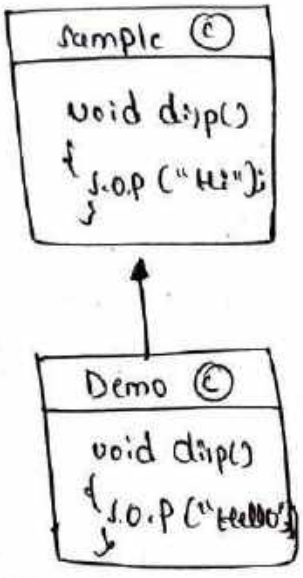
```
abstract interface sample2
    {
    public abstract void disp();
    public abstract void test();
    }


abstract class Demo2 implements sample2
    {
        //public abstract void disp();
        public void test()
            {
            S.O.P (" Hey this sub class");
            }
    }

    class samplera extends samplera
    {
        public void disp()
        {
            S.O.P(" Hey I am subclass 2");
        }
    }

class Main class
{
    public static void main (String[] args)
    {
        samplera raj= new samplera();
                raj. disp();
                raj. test();
    }
}
```

Realtime example :-

```
abstract interface puma
    {
    public abstract void shoes ();
    public abstract void Bags ();
    }

abstract interface Nike extends puma
    {   // public abstract void shoes ();
        public void bags()
        {s.o.p (" Nike bags");
        }
    }

class Rajoutlet implements Nike
    {
        public void shoes()
        {
        s.o.p ( " Jingilala shoes");
        }
        public void bags ()
        {s.o.p (" Jingilala Bags");
        }
    }

class Main class
{ public static void main (String [] args)
    {
        Rajoutlet. raj = new Rajoutlet();
                raj. shoes();
                raj. bags ();
    }
}
```

~~POLYMORPHISM~~ :-

② In the case of method overriding even those, if upcasted, we get overridden implementation.

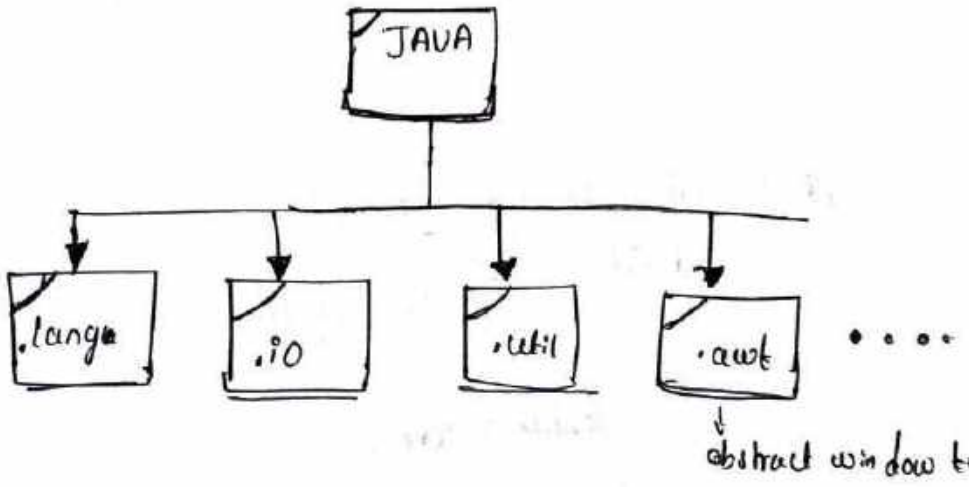



```
Sample s1 = new sample()
s1. disp();
```

21/11/2022

JAVA LIBRARIES :-

(*) Java libraries are the inbuilt folders, which consist of inbuilt class and interface.



Wrapper class :-

(*) Wrapper class are the inbuilt class. for each and every primitive data type, we have the corresponding class called wrapper class.

(*) Wrapper class belongs to [Java.lang] package.

(*) It's has two mechanism boxing and unboxing.

Boxing :-

(*) Converting from primitive data type to the corresponding wrapper class object. is called as boxing.

Unboxing:

(*) converting from wrapper class object back to the primitive data type is called as unboxing

(*) Boxing and unboxing happens implicitly.

~~Double d1 = new Double (50.6d)~~
~~d.(d1);~~
~~a1 object~~

~~double = d1;~~
~~double d1 = d;~~
~~d1.(a1);~~ ~~50.6d~~

Boxing:

~~unbo~~ boxing

```
Integer a1 = new Integer (10);
S.O.P(a1)                10 object
```

→ `int s = a1;`
S.O.P(s); 10

```
Double d1 = new Double (50.6d)
S.O.P(d1);

double d = d1;
S.O.P(d)   50.6d
```

PRIMITIVE DATA TYPE	WRAPPER CLASS
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Boxing

Unboxing

Integer

```
Integer o1 = new Integer (10);
S.O.P(o1)
```

boxing

o1 — object

```
int a = o1;
S.O.P(a)
```

unboxing

10

O/P

10

Character

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

Char ch = c;
S.O.P(ch);
```

A

O/P

'A'

Double

```
Double d1=new Double (50.6d);
    s.o.p(d1);

    double d = d1;
    s.o.p (d);
```

O/P

50.6d

Boolean

```
Boolean b=new Boolean(True);
    s.o.p. (b);

    boolean b1 = b;
    s.o.p (b1);
```

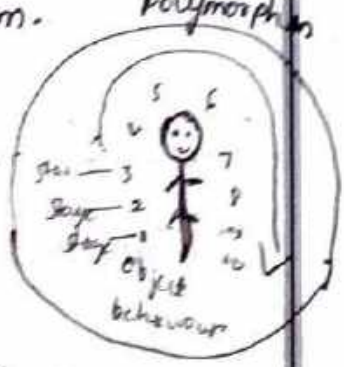
O/P

True.

22/11/2022

POLYMORPHISM :-

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



* Poly means "many" morphism means "forms"

* In polymorphism we have two types :-

1] Compile time polymorphism (Static polymorphism)

2] Run time polymorphism (Dynamic polymorphism).

1] Compile time polymorphism :-

* The method Declaration will get binded, to ~~each~~ its definition, at the compile time, by the compiler based on the argument passed is called as compile time polymorphism

* Since, the method declaration will get binded, to its definition, at the compile time. hence its called as early binding.

* once, the method declaration gets binded to its definition, it can't be rebinded Hence, it is called as static binding.

* Method overloading is an example for compile time polymorphism.

2] Run time polymorphism :-

* The method declaration will get binded to its definition. at the run time, by the JVM based on the object created is called as Run time polymorphism.

* Since, the method declaration ~~will~~ gets ~~find~~ binded to its definition. ~~at~~ at the run time Hence its called as late binding.

* Once, the method declaration gets binded to its definition it can be rebinded Hence, its called as dynamic ~~b~~ binding.

* method overriding is an example for run time polymorphism.

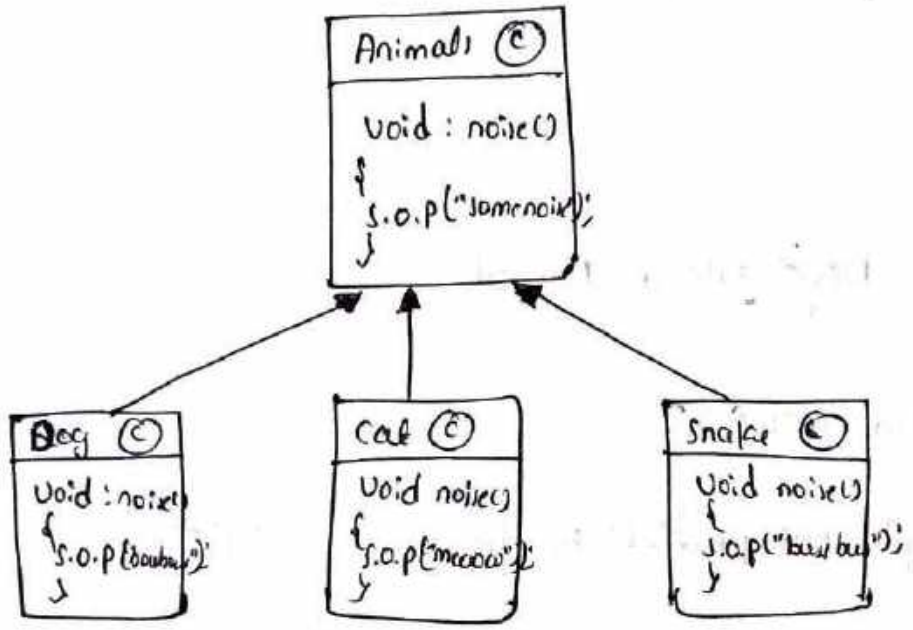
Assignment:

1) Write some polymorphism.

polymorphism flow diagram:

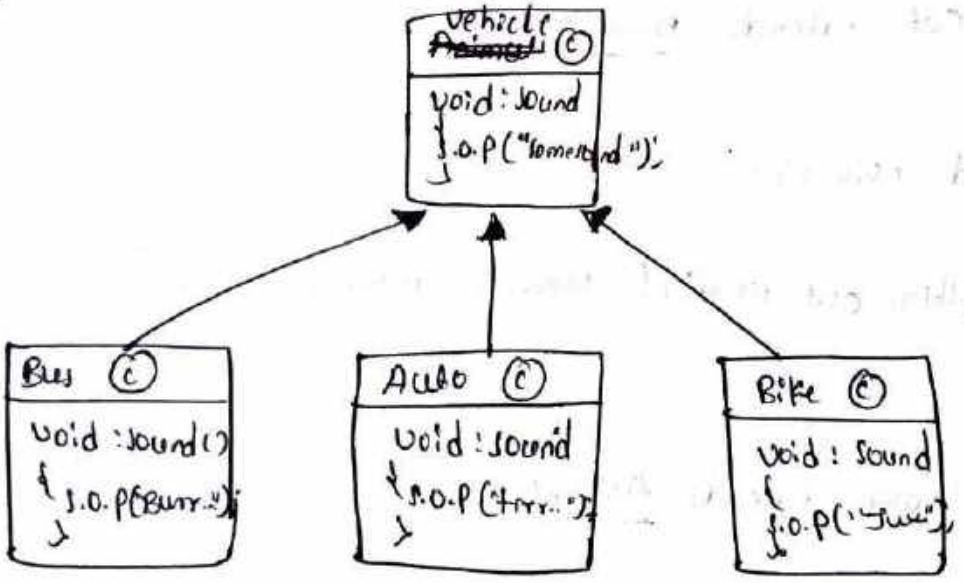
(1)

Animals



(2)

Vehicles

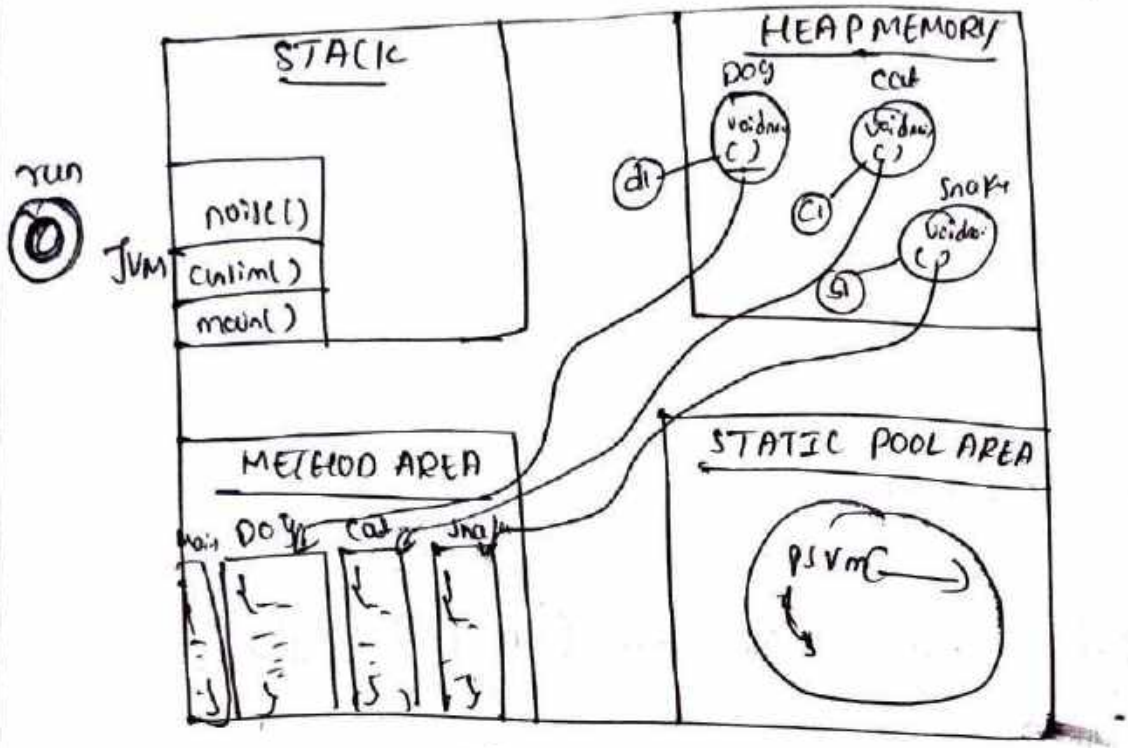


Program 1:

```
class Animal
{
    void noise() -
    {
        system.out.println("some noises"); -
    }
}

class Dog extends Animal
{
    void noise() -
    {
        System.out.println(" bo bow bow..."); -
    }
}

class cat extends Animal
{
    void noise() -
    {
        system.out.println(" Meaow meaow ..."); -
    }
}

class snake extends Animal
{
    void noise() -
    {
        system.out.println(" bw... buses... buss"); -
    }
}
```

```
class Simulator
{
    static void anim (Animal a1) = d1;
    {
        a1. noise ();
    }
}

class Main class
{
    public static void main (String [] args)
    {
        Dog d1 = new Dog ();
        cat c1 = new cat ();
        Snake s1 = new Snake();

        Simulator. anim (d1);
        Simulator. anim (c1);
        Simulator. anim (s1);
    }
}
```

JVM memory Diagram :-



JVM memory space :-

Why we go for polymorphism.

- generalization.
- single point of contact.
- loose coupling.
  - if any internal announcement happen it will not affect the usage.

23/11/2022

INSTANCE OF OPERATOR:

(*) It is an Operator which will return through object, belongs to the class hierarchical then it will return true, else it will return false.

```
class Simulator
{
  static void ani(Animal a1)
  {
    a1.noise();
  }
}
```

```
Class Simulator
{
  static void ani(Animal a1)
  {
    if(a1 instance of Animal)
    {
      a1.noise();
    }
  }
}
```

Example for compile time polymorphism :

compile time polymorphism :- The method declaration will get binded it definition at compile time by the compiler based on the Arugument passed is called as compile time polymorphism.

Example :- program :-

```
class Whatsapp
{
    void send (int no)
    {
        s.o.p ("The no is :" + no);
    }

    void send (int no , string name)
    {
        s.o.p ("The no & name" : + no + name);
    }

    void send (String name, int no)
    {
        s.o.p ("The name & no :" + name + no);
    }
}

class Main class
{
    Public static void main (String [] args)
    {
        Whatsapp w = new Whatsapp ();
            w. send (11);
            w. send (11, "NAGU");
            w. send ("NAGU", 11);
    }
}
```

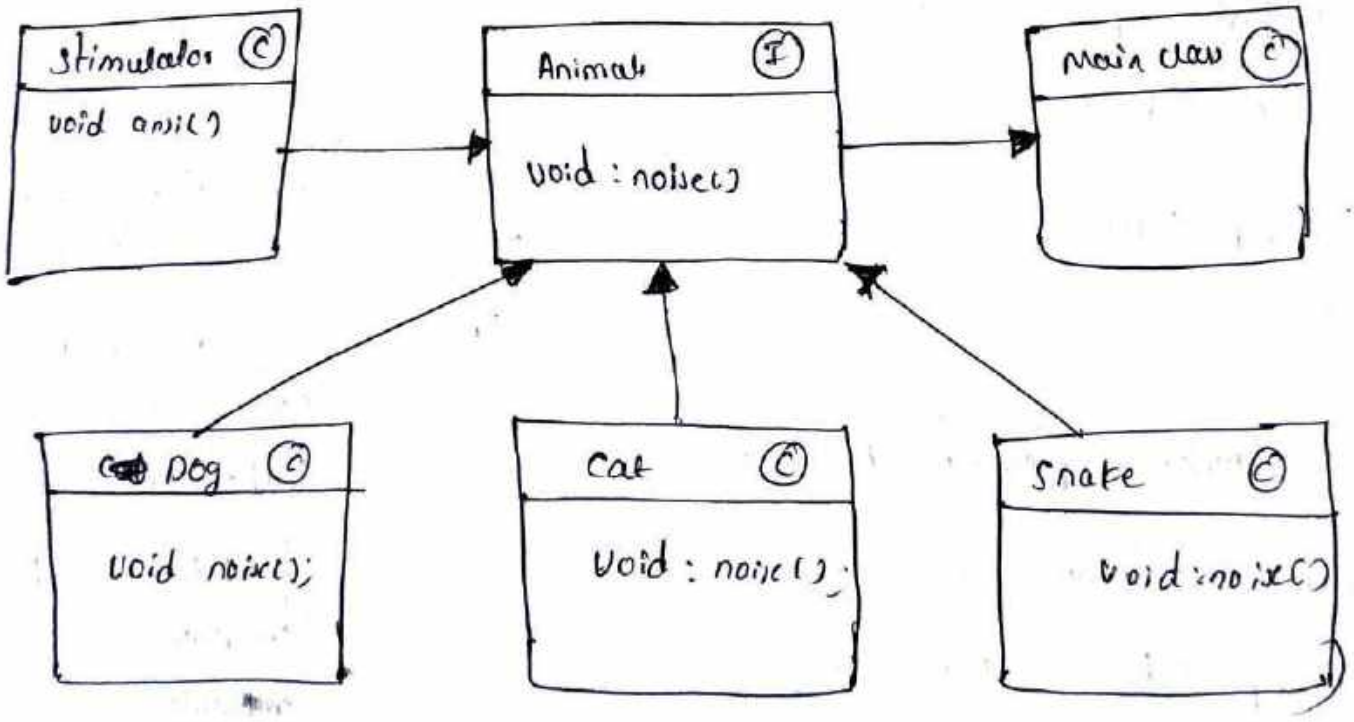
O/P

The NO is :11
The NO & name :11 NAGU
The name & NO is :
NAGU 11

ABSTRACTION :-

" Hiding the complizity of system, to exposing user required functionality to the end users is called as abstraction."

- abstraction is hides the user required functions.
- through Interface we can achive 100% abstraction.
- through abstract class we can achive up to 100% abstract.
- create a reference variable of interface type and initilize that reference variable with the implementation class object. this is how we achive abstraction.
- user friendlyness and security.



Program :

```
Interface Animals
{
    void noise ();

}
class Dog implements Animals
{
  public void noise ()
  {
    S.O.P ("Bow... Bow...");

  }
}

class Cat implements Animals
{
  public void noise()
  {
    S.O.P ("Meaow... Meaow...");
  }
}
class Snake implements Animals
{
  public void noise()
  {
    S.O.P ("Boos... Booos...");
  }
}
```

```
class Stimulator
{
  ~~void~~ static void ani
        (Animals a)
  {
    a.noise();

  }
}

class Main class
{
  public static void main
        (Strings args)
  {
  Dog d= new Dog();
  Cat c= new Cat();
  Snake s=new Snake();

  Stimulator.ani (d);
  Stimulator.ani (c);
  Stimulator.ani (s);
  }
}
```

O/P

```
Bow... Bow..
Meaow.. Meaow..
Boos ... Booos.
```

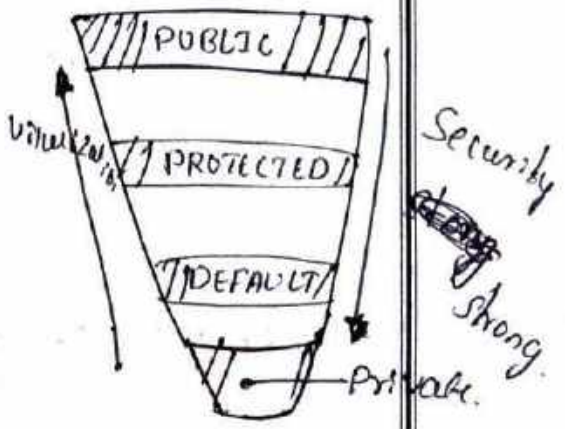
ACCESS SPECIFIER'S :-

Access specifier :-

(*) Access specifier it is used to ~~restrict~~ restrict the access from one class to another class. is called as Access specifier.

(*) In access specifier we have 4 types

1] PUBLIC
2] PRIVATE
3] PROTECTED
4] DEFALUT / PACKAGE LEVEL



1] PUBLIC :- Any member which is declared with the keyword is public is called as public access specifier.

(*) it can be access within the class.
(*) it can be access within the package.
(*) it can be Any way.

2] PRIVATE :- Any member which is Declared with the keyword Private is called as private access specifier.

(*) private can be access within in the class.

3) PROTECTED :- Any member which is declared with keyword PROTECTED is called as Protected access specifier.

- (1) It can be access within the class.
- (2) it can be access within the package
- (3) It can be access outside the package
- (4) it can be access ~~inside~~ outside the package and within the package with this relationship.

4) DEFAULT / PACKAGE LEVEL :- Any member which is not declared ~~with~~ Any keyword is called Default access specifier.

- (1) It can be access within the class.
- (2) it can

25/11/2022

Access specifier

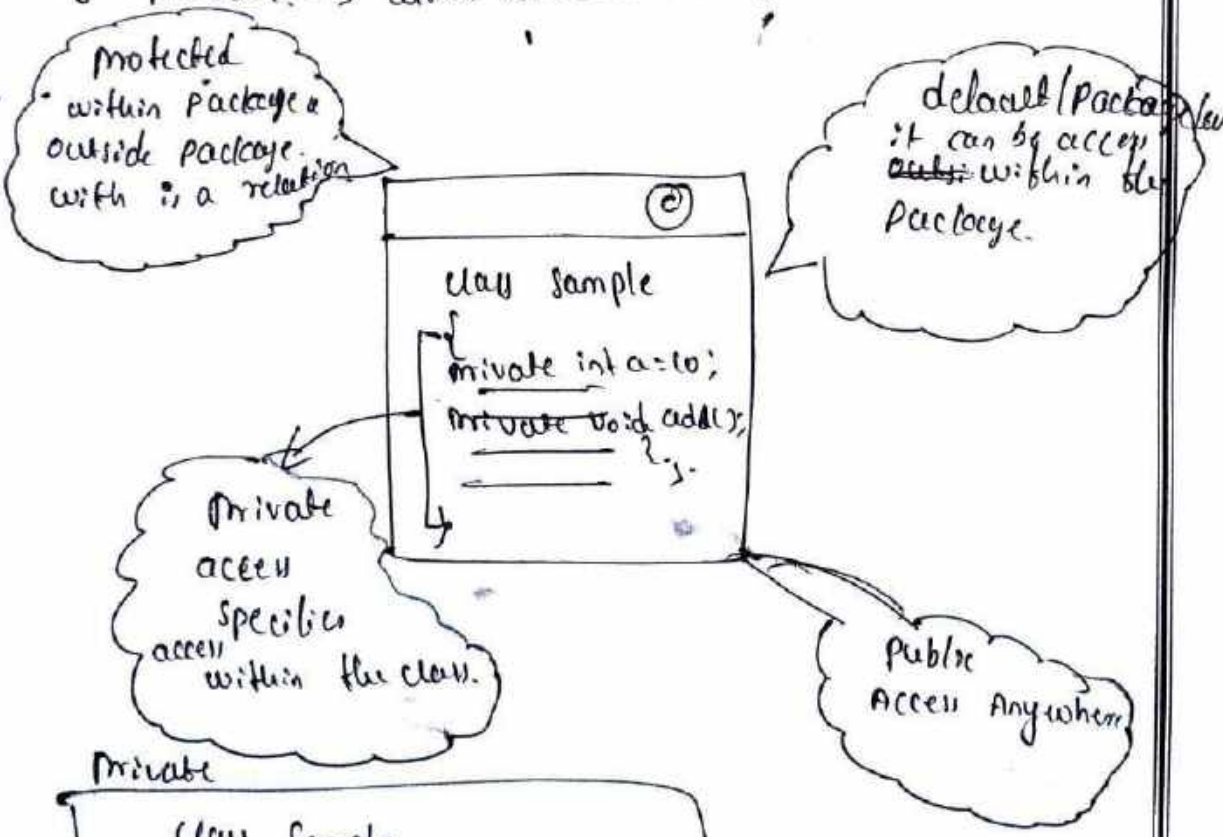
Access specifier	Inside class	Inside package	Outside package
Public	yes	yes	Yes
protected	yes	yes	Yes
default	yes	yes	NO
private	yes	NO	NO

Access specifier :-

(*) It is a access permission / specifier the class from one class to another class. is called Access specifier.

It have 4 types :-

1) Public :- Any member which has declared with the keyword public that is public access specifier.
2) protected :- " " keyword protected. that is protected access specifier.
3) default (package level - No keyword mention within package.
4) private → within the class access.



private

```
class Sample
    private int a = 10;
    private void display
    {
    }
}
```

Default

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

Protected

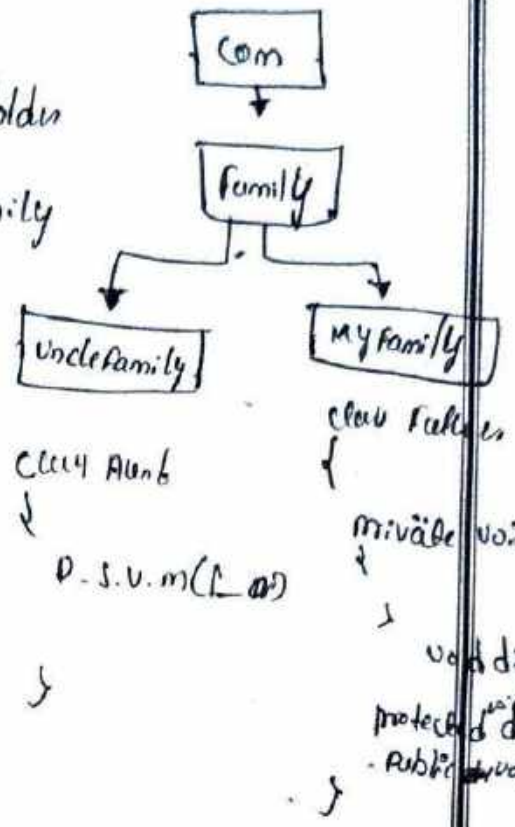
```
class sample
{
    protected int a=10;
    protected void disp()
    {
    }
}
```

Public

```
class Sample
{
    public int a=10;
    public void disp()
    {
    }
}
```

Access specifier :- program :-

Com is a folder it have one more folder family that contain sub folder myfamily myfamily inside father class.



- first stmt: // package is a folder

```
package stmt;
```

second stmt:

```
import stmt.
```

import is a keyword for importing files into a folder with virtually.

```
package com.family.myfamily.
```

```
import com.family.unclefamily.father.
```

Program:

```
Package com.family.myfamily;
import com.family.unclefamily.father;

public class Father
{
    private void atm()
    {
        s.o.p("Father Atm");
    }

    void car()
    {
        s.o.p("Father car");
    }

    protected void bike()
    {
        s.o.p("Father bike");
    }

    public void cycle()
    {
        s.o.p("Father cycle");
    }
}

package com.family.myfamily;
public class Son
{
    Father f=new Father();
        // f.atm();
        f.car();
        f.bike();
        f.cycle();
}
```

```
package com.family.unclefamily;
import com.family.myfamily.father;

public class Aunty extends father
{   public static void main(String[] args)
    {   Aunty a = new Aunty();
        // a.alm();
        // a.car();
           a.bike();
           a.bycle();
    }
}
```

① O/P

father car
father bike

② O/P

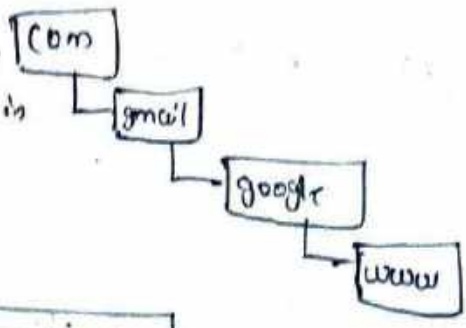
father bike
father bycle.

PACKAGE :-

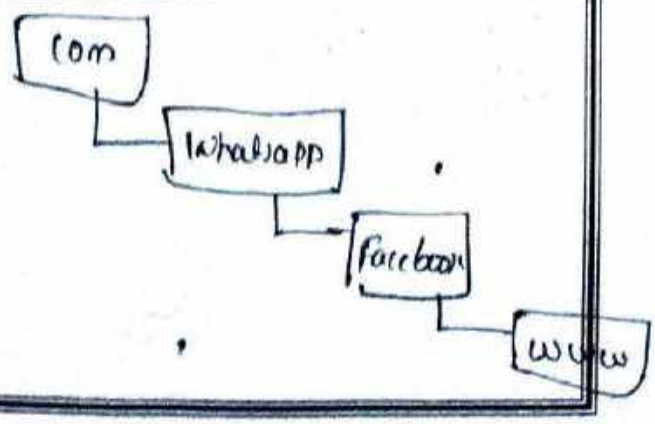
- Package is a folder structure which is used to store similar kind of files.
- In any Java file, The package stmt should be the first stmt. second should be import stmt. The Third stmt can be any Java Type.
- Import stmt can be more than one stmt.
- The package structure should be the reverse order of the URL

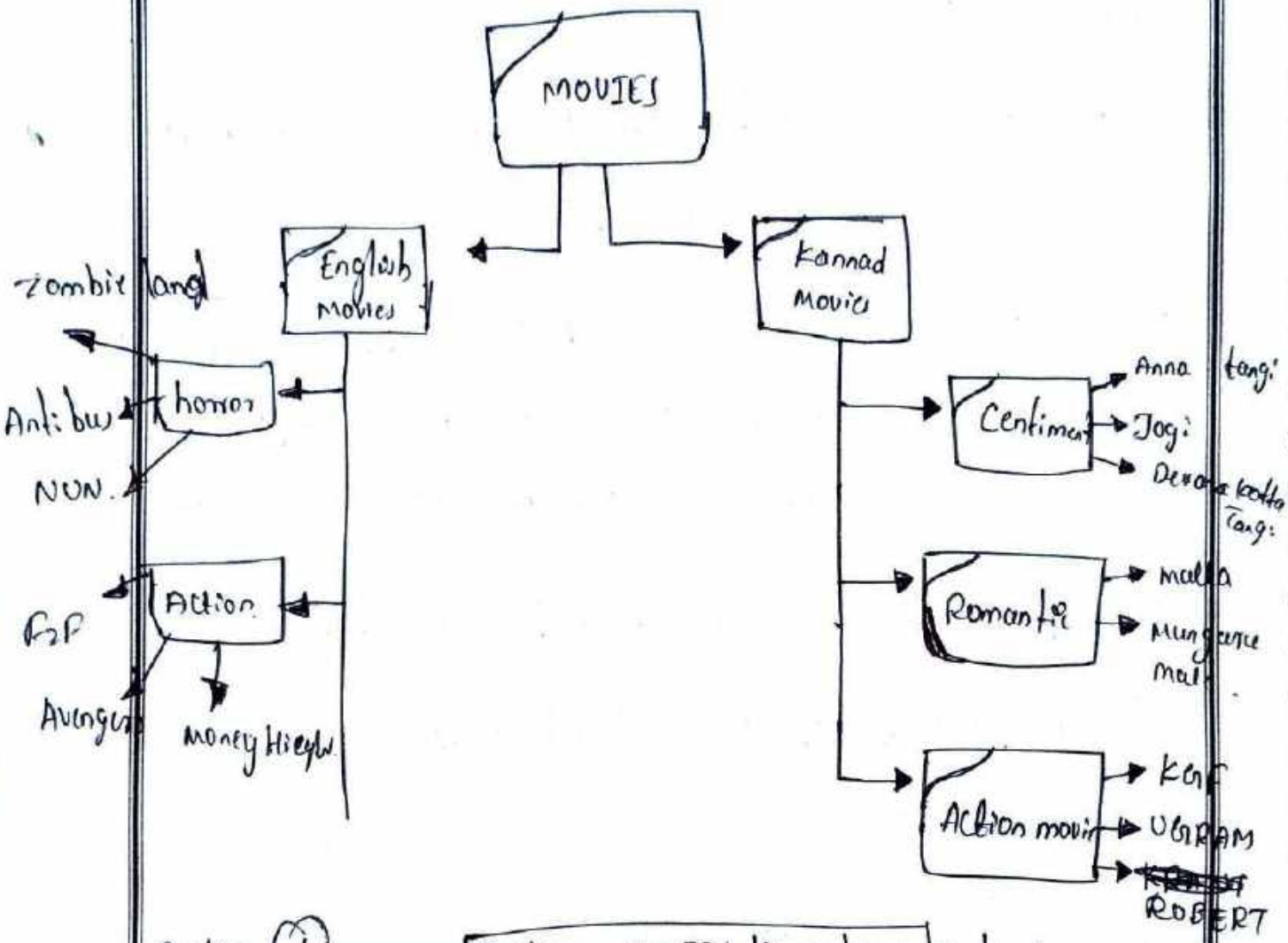
(1) www.google.gmail.com

// reverse order of folder we should create in eclipse IDE.



(2) www.facebook.whatsapp.com





Package stmt (1) — package MOVIES.kannadmovie.centiment.Annatangi

import stmt (2) — import MOVIES.kannadmovies.centiment.Jogi
import MOVIES.kannadmovies.Romantic.malla
import MOVIES.kannadmovie.Action movie.ROBERT

Java Type (3) — class Hybrid movie
{
}

any class, interface
enum, annotation

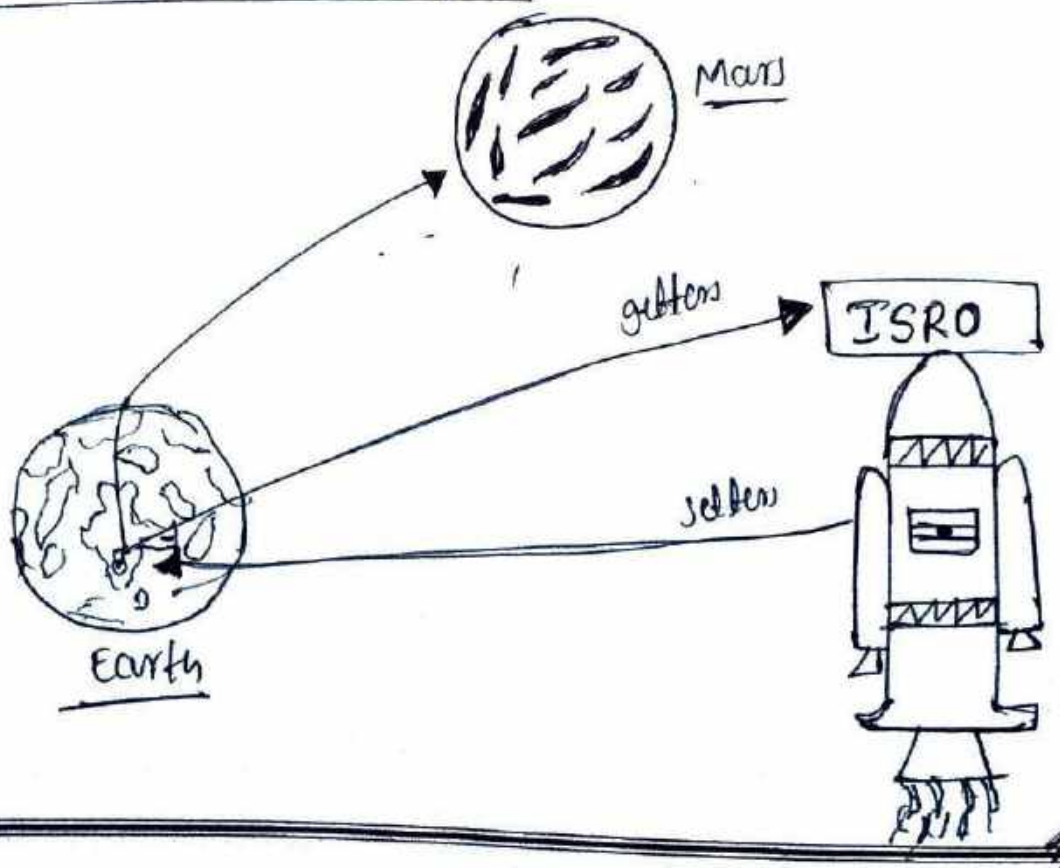
28/11/2022

ENCAPSULATION :-

- Encapsulation is a one of the oop's concepts.
- Encapsulation is by default Java is a Encapsulation.
- We can't Declared Any variable outside the class.
- We can't print any stmt outside the class.

" Declare data members as a private and restrict the ~~Dirct~~ Direct access outside class, and provide the indirect access through Public services is called getters and setters is called as ENCAPSULATION "

Encapsulation flow diagram :



JAVA BEAN CLASS :- Same as Encapsulation.

(R) "Declare data members as private, and Restrict the Direct access from outside class. and Indirect access through public services is getters and setters is called JAVA BEAN CLASS."

Or

(r) Rapping a bul data members and function member is called as Java bean class.

purpose :- get ATM Password, Get ATM password, Instagram,

(1) Security

Program on Encapsulation :-

```
class Encaps
{
    private int a=10;
    public int getA()
    {
        return a;
    }
    public void setA(int a)
    {
        this.a = a;
    }
}
class Main class
{
    public static void main (String[] args)
    {
        Encaps e = new Encaps();
        S.O.P (e.a);
        e.setA(20);
        S.O.P (e.getA());
    }
}
```

O/P

10
20

```
class Rocket_Iro
{
    private string str = "MONEY";

    public string getA()
    {
        return str;
    }

    public void setA(string str)
    {
        this.str = str;
    }
}

class Main class
{
    public static void main(String[] args)
    {
        Rocket_Iro i = new Rocket_Iro();
        System.out.println(i.str);
        i.setA("ROCKET");
        System.out.println(i.getA());
    }
}
```

O/P

MONEY
ROCKET

29/11/2022

OBJECT CLASS :-

- object class is supermost class in java.
- object class belongs to java.lang package.
- each and every class extends object class.

```
class [object] implicitly
{
  ...
}
```

- OBJECT CLASS have following methods :-

1] int hashCode() → // it will generate hashnumber unit digit code.
2] boolean equals() → // to compare object address.
3] object clone()
4] String toString() → // generate fully qualify path.
5] void notify()
6] void notifyAll()
7] void wait()
8] void finalize()

NOTE :- Object class ~~All~~ All methods in we will use CONSTRUCTOR

14 To String Method :-

- To string method is a non-static, non final method of object class.
- To string method will be inheriting to each and every class.
- whenever, to print the reference variable to string method will get invoked where it will return fully qualified path in the form of string.
- To string method can be invoked implicity and explicity.
- The signature of the string method is public string to string

  public string to string

- we can override the to string method.

Example pgm :-

```
package objectTopic;
public class Demo1 extends object
    Public static void main (String[] args)
        Demo1 d=new Demo1 ();
        S.O.P (d.toString ());
```

O/P

objectTopic.Demo1@ 1452[illegible]

Package class name Hexa Decimal

Program :-

```
package objectTopic;

class Demo extends object
{
    public string toString ()
    {
        return "Hello";
    }

    public static void main (String[] args)
    {
        Demo d1=new Demo ();
        S.O.P (d1.toString());
    }
}
```

O/P

```
Hello
```

```
package objectoptic;

class sample extends object
{
    public string to string ()
    {
        return " Hi JAVA" ;

    }

    public static void main (string [] args)
    {
        sample s= new sample ();
        s.o.p ( s. to string ());
    }
}
```

O/P

```
Hi JAVA
```

Real time example on to string method (1) :

1] Write a program to override to string method to display student details.

steps of program :

- consider to attributes studentId, studentname.
- create constructor and initilize the attributes on creating object.
- override to string method where it returns current object to attribute values.

(1) student Id,

(2) student name.

Program: Student details

```
class student
{
int stu-id;
String stu-name;

    Student (int stu-id , String stu-name)
    {
      this. stu-id= stu-id ;
      this. stu-name= stu-name;
    }

  public string to string ()
  {
    return " student id :" + " this. stu-id + " student
                                                  NAGU
            name :" + stu-name ;
  }


public static void main (String [] args)
{
    Student s= new student (11, "NAGU");
    s.o.p ( s. to string ());

    Student s2 = new student (12, "NAVEEN");
    s.o.p (s2. to string ());
}
}
```

O/P

The student id:11, The student name is NAGU

The student id :12 The student name is NAVEEN

Employee Detail :-

```
class Employee
{
  int emp-id;
  String emp-name;
  int emp-sal;
  Employee ( int emp-id , String emp-name, int emp-sal)
  {
    this. emp-id = emp-id;
    this. emp-name= emp-name;
    this . emp-sal = emp-sal;
  }
  public String to string ()
  {
  return "the employee id is:"+ emp-id + "The employee name is:"
          + emp-name+" the employee salary is:" + emp-sal ;
  }
public static void main (String [] args)
 {
    Employee e= new Employee (11, "NAEIU", 20,000);
    S.O.P( e. to string());

 }
}
```

O/P

```
the employee id is: 11 The employee name is: NAEIU
the employee salary is: 20,000
```

② Boolean Equals Method :- //object address.

- Equals method is a non-static non final method of object class.
- Equals method it is inheriting to each and every class.
- Equals method will compare object address only.
- if the object address are same it will return the true result, else it will return false result.
- the return type of equals method is "boolean".
- Equals method signature is

```
public boolean equals ();
```

Program :

WAP to override Equals method

```
package objectTopic;
class Sample extends object.
{
    public static void main (String[] args)
    {
        Sample s = new Sample ();      // 1st object
        Sample2 s1 = new Sample2 ();   // 2nd object
        S.O.P ( s . equals (s1) );     // s = s1
                                       o/p
                                       FALSE
    }
}
```

Package object topic

```
class Demo extends object
{
    public static void main (String[] args)
    {
        Demo d = new Demo ( );      // #11 -> d address
        Demo d2 = d;                // #20 -> d2 address
                                    d2 = d  // = //
        S.o.p ( d. equals (d2));
    }
}
```

O/P

TRUE.

Types of to override the Equals methods.

① consider the attributes (variables) through the constructor one creating the objects.

② Override equals methods before that upcasting and downcasting " Comparision logic.

Write a program to compare to employee salary are same.

```
class Employee extends object
{
    int emp_id;
    String emp_name;
    int emp_sal;
    Employee (int emp_id, String emp_name, int emp_sal)
    {
        this. emp_id = emp_id;
        this. emp_name = emp_name;
        this. emp_sal = emp_sal;
    }
    public boolean equals (Object obj) //upcasting.
    {
        Employee e1=(Employee) obj; //down casting
        return this emp_sal == e1. empsal; //Comparision.
    }
    public static void main (String[] args)
    {
        Employee e = new Employee (1, "NAGU", 25,000);
        Employee e1 = new Employee (2, "NAGU", 25,000);
        S.O.P ( e. equals (e1));
    }
}
```

O/P

TRUE

7/12/2022

Hash code ():-

- Hash code method is non-static, non final method of object class.
- Each and every class extends object class.
- whenever an Hash code method is invoked it will generate a unic integer number called hash number based on the object address.
- The Hash number will be generated by using the Hashing Algorithm.
- The signature of hash code method is public int hashCode()
- We can override hash code method.

Syntax:

```
public int hashCode()
```

" Hash Number will generate based on object address is called hash code "

program:

```
package objectTopics

public class Demo1 extends object
{
    public static void main (String[] args)
    {
        Demo1 d1 = new Demo1();
        S.O.P (d1.hashCode());

        Demo1 d2 = new Demo1();
        S.O.P (d2.hashCode());
    }
}
```

output

```
2018699554
1311053135
```

```
package objectopic ;

public class Demo2
{
    public int hasCode ()
    {
        return 123;
    }
    public static void main (String [] args)
    {
        Demo2 d1 = new Demo2 ();
        S.O.P (d1. hasCode ());
    }
}
```

Output

123

STRING CLASS :-

8/12/2022

- Any class which is declared with keyword final is called final class.
- final class can't be inherited but, we can create object.
- Any method which is declared with the keyword final is called as final method.
- final method can be inherited but ~~it not~~ we can't override.
- we can overload method.
- we can declare a class, variable and method as final, but not be constructor.

```
final class Demo
{
    void disp()
    {...}
}
```

① we can't inherit final class

```
class Demo1
{
    final void disp()
    {
    }
}
```

② we can inherit final method.

```
class Demo2
{
    final void disp()
    {}
    final void disp(int)
    {}
}
```

③ we can inherit final method with parameters.

STRING CLASS

- string class is a final class which belongs to java.lang package.
- string class can't be inherited, but we can create object.
- string objects can be create ~~with~~ in two ways

  1) `string s1 = "Hello";`
  - string object without new operator.

  2) `string s2 = new String("Hi");`
  - string object with new operator.

- All the string object will get stored in string pool area.
- string pool area is divided into two parts.
  1) Constant pool area
  2) Non-constant pool area
- All the string objects which is created without new operator will get stored in constant pool area.
- All the string objects which is created ~~new~~ with new operator will get stored in non-constant pool area.

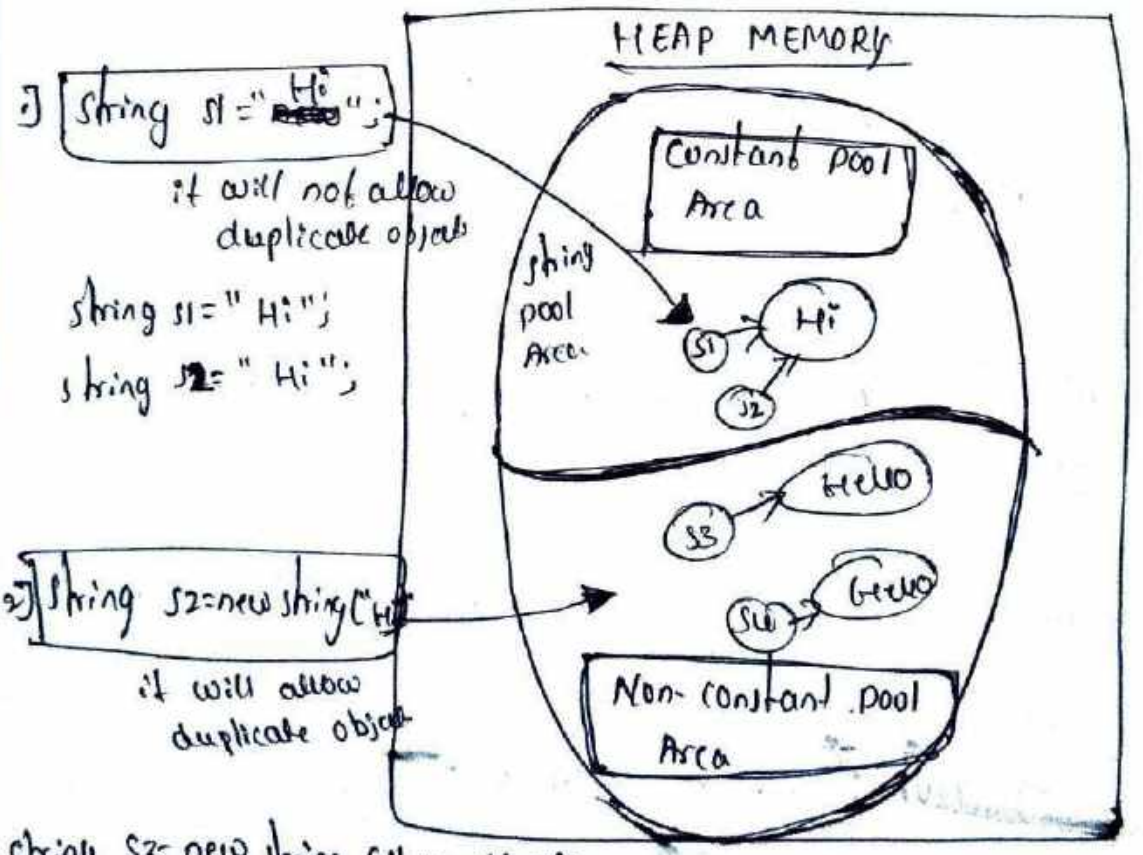
* string object without new operator will not allowed duplicate objects.

* string object with new operator will allowed duplicate objects.

STRING CLASS :- Two way object creation

1] string s1 = "new";

2] string s2 = new string ("Hi");



string s3 = new string ("Hello");

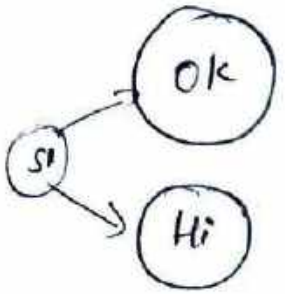
string s4 = new string ("Hello");

STRING CLASS IMMUTABLE :

What

IMMUTABLE :- immutable means it will not change state of the object.

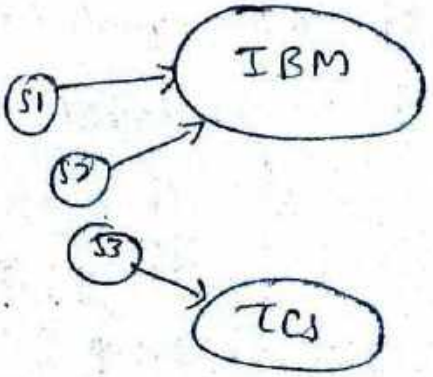
```
String s1 = "ok";
       s1 = "Hi";
```



why

① When multiple reference variable a pointing to single object and if one of the reference variable ~~re-initiate~~ re-initilize, it will not affect other reference variable. This why String class is immutable.

```
string s1 = "IBM";
string s2 = "IBM";
string s3 = "IBM";
       s3 = "TCS";
```



① Any operation perform on a string with get stored in non-static pool area.

EQUAL OPERATOR (==) :-

① it is an a operator with represents (==).

② it's comparision operator, it compares operation.

③ it will compare object address only.

```
String s1 = "ok";
String s2 = "ok";
```

s1 == s2

True

same address ok

s1 == s2 → True

④ it cannot be overriden.

EQUAL METHOD equals() :-

① it's is a non-static method equals().

② it will compares object values.

③ it will equals object values then that is true.

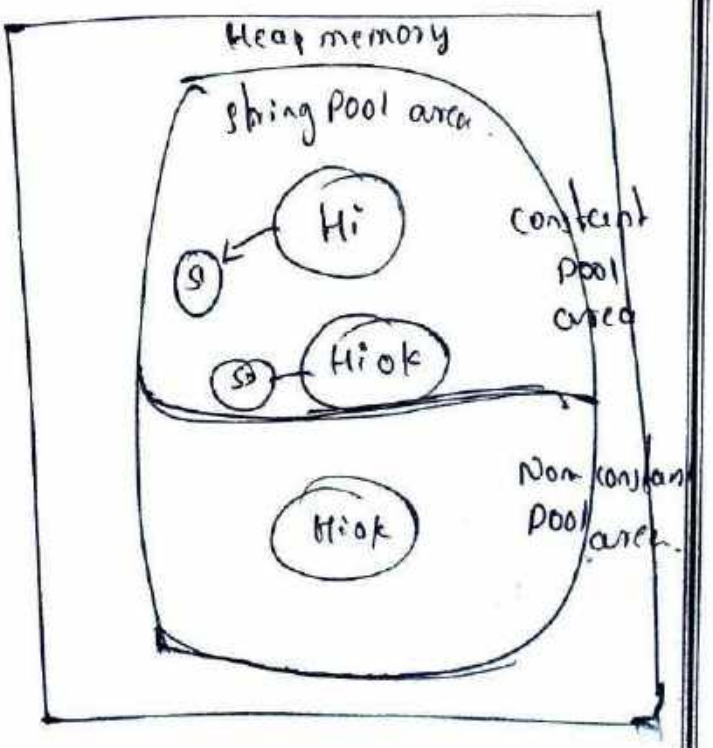
it is not equal object values then that is not true body.

```
String s1 = "Hi";
String s2 = "Hi";
S.O.P ( s1.equals(s2) ); // True
```

```
String s1 = "Hi";
s1 = s1 + "ok"
Hiok
```

```
String s2 = "Hiok
```

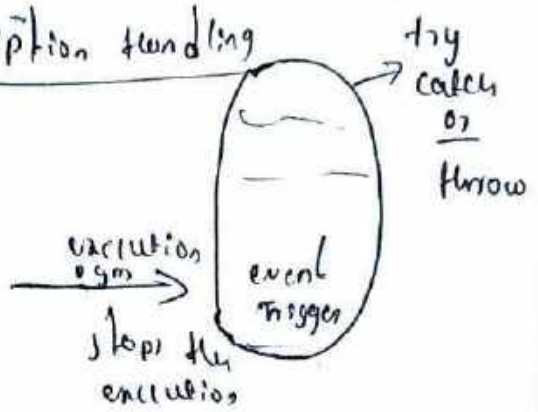
```
s.o.p(s1 == s2);
```



EXCEPTION HANDLING.

9/12/2022

" If is an Event Trigger, during execution of the program, which interupt execution and stops the execution, abnormally or ~~suddnel~~ suddenly Handling this event by using try, catch or throw is called Exception Handling "



SYNTAX :-

①
```
try {
    ____
    ____
    ____
    ____
  }
catch (Exceptionclass ref variable)
  {
    ____
    ____
    ____
    ____
  }
```

(2)

```
try {

    }
catch (        )
{

    }
catch (      )
{

    }
```

(3) Nested try catch :-

```
try {
    try {
        }
    catch (  )
    {
    }
}
catch (   )
{
}
```

4]
```
try {

}
catch (    )
{
    try {
    }
    catch ( )
    {
    }
}
```

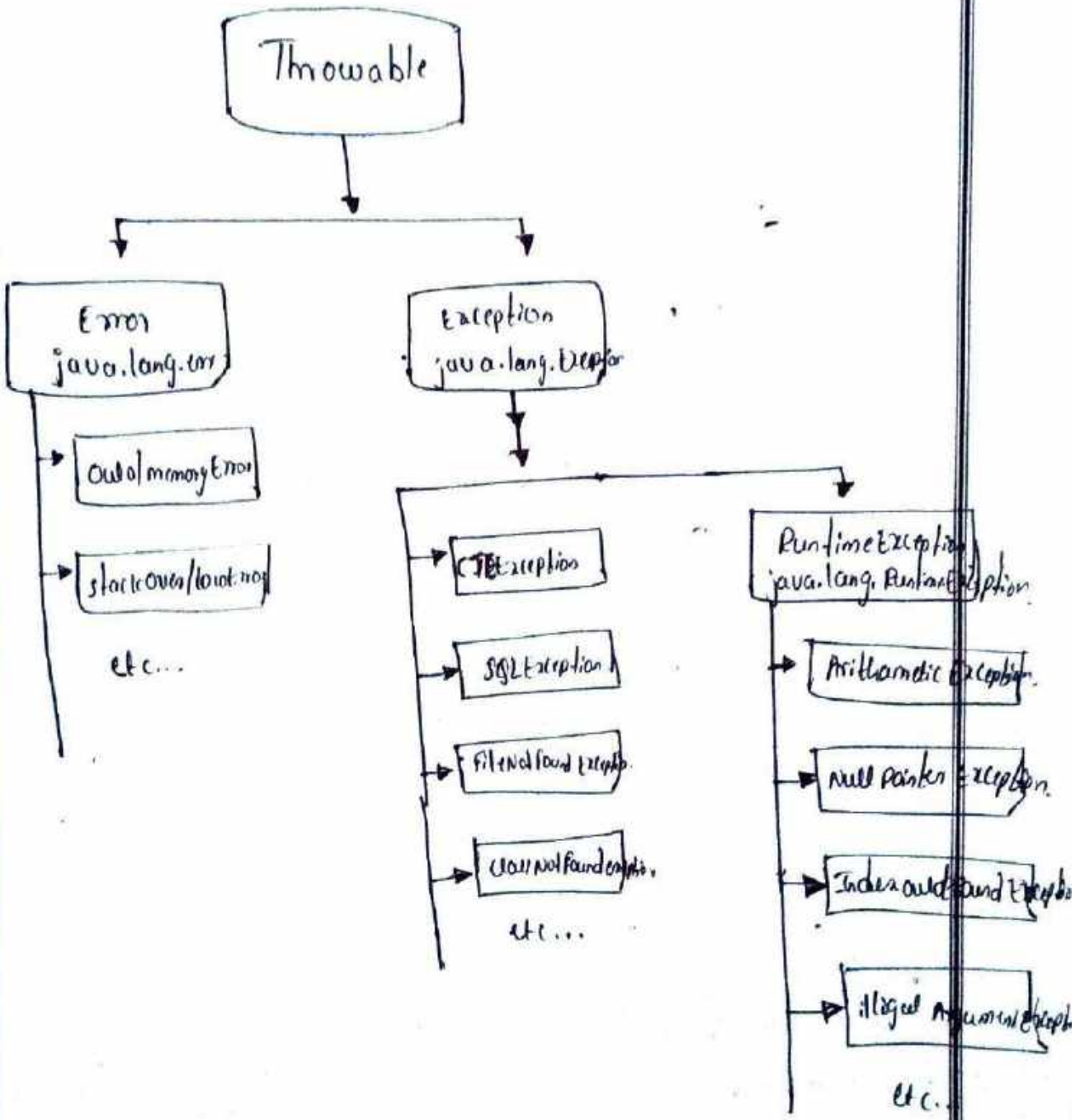
(5)
```
try {
}
finally (    )
{
}
```

5]
```
try {
}
finally {
}
```

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

(6)

Exception Handling diagram :-



:- RUNTIME EXCEPTION :-

an exception occured during run time called Runtime exception

Arithamatic Exception

programs:

```
class Exceptiopic
{
    public static void main (String[] args)
    {
        s.o.p ("Main starts---");
        try
        {
            int i = 1/0;
        }
        catch (ArithamaticException e)
        {
            s.o.p ("Handled");
        }
        s.o.p ("Main Ends");
    }
}
```

o/p

```
--- Main starts ---
Handled
--- Main Ends ---
```

Array out of bound exception:

```
package Excep

class Sample
{
    public static void main (String[] args)
    {
        S.o.p ("--- Main starts ----");
        int[] arr={ 10, 20, 30 };
        try{
            S.o.p ( arr[9]);
        }
        catch (ArrayIndexOutBoundsException e)
        {
            S.o.p ("caught");
        }
        S.o.p (" *** Main Ends *** ");
    }
}
```

O/P

```
-- Main starts ---
caught
*** Main Ends ***
```

12/12/2022

2] Exception Handing :

2]
```
try
{
  ___
  ___
  ___
}
catch(    )
{
  ___
  ___
  ___
}
catch(    )
{
  ___
  ___
}
```

(*) try block have multiple catch

(*) try block have there is an exception code, Other ~~to~~ catch blocks will manage if based exception Or error.

```
class Demo
{
Public static void main(String[] args)
{
  try
  { int a = 1/0;
  }
  catch (ArithmaticException e)
  {
  ~~S.O.P (e)~~ S.O.P ("managed");
  }
  catch (ArrayIndexOutofBoundsException e)
  {
    S.O.P ("caught");
  }
}
}
```

WAP To Develop multiple catch blocks

```
class Excep
{
    Public static void main (String[] args)
    {   S.O.P ("MAIN START");
        try
        {
            int a = 1/0;
        }
        catch (ArrayIndexOutOfBound Exception e)
        {
            S.O.P ("Manged");
        }
        catch (Arithametic Exception e)
        {
            S.O.P ("Handled");
        }
        S.O.P ("MAIN END");
    }
}
```

O/P

```
MAIN START
Handled
MAIN END
```

3] Nested try catch :

```
try
{
    try
    {
      ==
      ==
    }
    catch (Exception referenvariablename)
    {
      ==
      ==
    }

}
catch (Exception referen variablename)
{
    ____
    ____
    ____

}
```

```
try
{
    try{
      } =
    catch ( )
      }
}
catch (      )
}
```

Nested try catch blocks :

there is no exception.

```
class sample
{
  public static void main (String[] args)
  {
        int a = 8/2; // = 4

        int[] arr = { 1, 2, 3, 4, 5 };
        s.o.p ( arr[a]);
  }
}
```

```
class sample
{
  public static void main (String[] args)
  {
      try {
        int i = 18/2; // 9
          try {
            int[] arr = { 1, 2, 3, 4, 5 };
          }
          catch (ArrayIndexOutBoundsException e)
          {
            s.o.p ("managed");
          }
      }
      catch (Arithamatic Exception e)
      {
          s.o.p ("Handled");
      }
  }
}
```

O/P

Handled

Exception Handling :

ii)

```
try {
    ==
    ==
}
catch (          )
{
        try {
            ==
        }
        catch (    )
        {
            ==
        }
}
```

Write a program to Demonstrate try catch inside one more try catch blocks

```
class Demo
{
    public static void main (String [] args)
    {
        try {
            int a = 1/0;
        }
        catch (ArithmaticException e)
        {
            S.O.P ("Handled");
            int [] arr = {10, 20, 30, 40};
            try {
                S.O.P (arr[8])
            }
            catch (ArrayIndexOutOfBoundsException e1)
            {
                S.O.P ("caught");
            }
    }
}
```

O/P

Handled
caught.

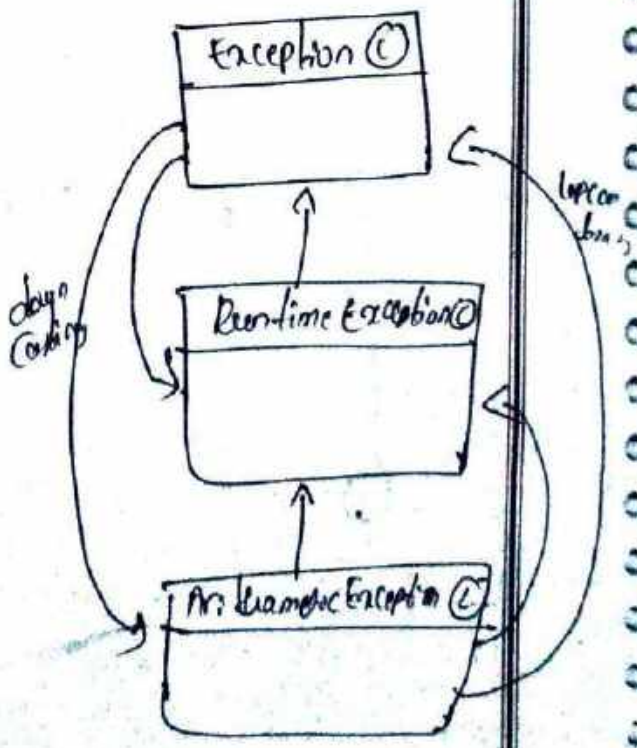
throw :- throw is used create exception.

throw new ArithmeticException ("/zero");

```
class sample
{
    public static void main (String[] args)
    {
        try {
            int i = 1/0 ;//throw new ArithmeticException("zero");
        }
        catch (ArithmeticException e) = new ArithmeticException();
        {
        }
    }
}
```

upcasting ↑

down casting ↓



* Once the Exception is address, if we try to re-address we get compile time error.

```
class Sample
{
  public static void main (String[] args)
  {
      try{
          int i = 1/0;
        }
      catch (Exception e)
      {
        S.o.p ("handled");
      }
      catch (ArithmeticException e)
      {
        S.o.p ("caught");
      }
  }
}
```

14/12/2022

Write a program to demonstrate try finally and try catch finally

finally block :

① finally block, block is exception Handling which will get executed mandatorily even those if exception is addressed or not addressed.

② we can develop finally block in try block and try catch block.

```
class Demo                                  // try catch finally block
{
    P.s.v.m (S[] a)
    {
        try {
            int a = 1/0;
        }
        catch (ArithmaticException e)
        {
            S.o.p ("Handled");
        }
        finally {
            S.o.p (" Hey I am finally block");
        }
    }
}
```

O/P

Handled
Hey I am finally block

try finally block :-

```
class Demo
{
  public static void main (String[] args)
  {
     S.O.P(" *** MAIN START) *** ");
       try{
          int a= 1/0;
          }
       finally {
          S.O.P(" Hey I am finally block");
          }
     S.O.P (" *** MAIN ENDS *** ");
  }
}
```

O/P

```
*** MAIN START) ***
Hey I am finally block
Exception in thread "main" java.lang.ArithmeticException: /by zero
        at excepl.Demo.main (Demo.java 6).
```

Null pointer Exception :-

```
class Demo
{
  public static void main (String[] args)
  {
    S.o.p("** Main starts **");
  // Demo d= new Demo();
  // S.o.p (d.hashCode());

    Demo d = null
  try{
    S.o.p (d.hashCode());
    }
    Catch (NullPointerException e)
    {
      S.o.p("Handled");
    }
    S.o.p ("** Main ends **");
  }
}
```

STACK WINDING :-

(*) When an exception occurs in an method which will propagate into the main method intern to the JVM, JVM doesn't have any handler and state will get destroyed. Hence, it is called as stack winding.

PRINTSTACKTRACE() :-

(*) It is a non-static method of throwable class which will be inherited to each and every class. which will prints the complete back space.

system.err

```
class system.
{
  static thing ini;
}
```

```
class Demo
{
 static void disp3() // throw propagate the method
 {  int a= 1/0; ------> // throw create exception
 }
 static void disp2()
 {  disp3();
 }
 static void disp1()
 {
    disp2();
 }
 p.s.v.m (s[] a)
 {
   try{
     disp1();
   }
   catch (ArithmeticException e)
   {S.o.p("Handle");
   }
}
```

O/P

Handled

EXCEPTION HANDLING :- 15/12/2022

" It is an a event trigger during execution of the program which has Interrupt occur stops execution. Abbotrly or suddenly, by handling this event using try, catch and throw is called as Exception Handling ".

- Exception handling have handeled in 2 ways
  1] try
  2] catch

- All the ~~abb. abr~~ abnormal stmt should be developed in the try block. and address them in the catch block.

- we cann't develop any stmt b/w try and catch block.

- we can't develop try block alone it will throw CTE

- for one try block we can develop one or more catch blocks.

- All the exception can be addressed by exception class.

- All the exception class is belongs to java.lang package

Exception are two types :-

1] Compile time Exception (checked Exception) :-

(A) Any exception which is occured at the compile time is called as compile time exception

(A) compile time exception can be handled by using try, catch, throw.

(A) it should be addressed at compile time itself.

2] Run time Exception (unchecked exception) :-

(A) Any exception which is occured at Run time exception is called run time exception.

(A) Run time exception can be handled by using try, & catch or throws.

(A) it should be addressed at run time itself.

(A) once the Exception is addressed we can't re addressed it ~~will~~ will throw CTE.

16/12/2022

Differentiate b/w Throw & Throws :

Throw	Throws
① Throw is used to create the exception	① Throws is used to propagate the exception
② throw should develop under method body	② throws should develop method declaration
③ throw can throw only one object at time	③ throws can propagate more than one exception

Assignment

write 1 pgm to demonstrate fo checked exception or user defined exception.

① Amazon
② OSA.
③ whatsapp
④ Facebook.
⑤ LinkedIn.

-: COMPILE TIME EXCEPTION :-

"compile time exception, when an exception occurred during compiliation time that is called compile time exception."

In compile time exception we will create our own Exception

Ex: shaadi.dot.com

```
class Shaadi.dot_com
{
  static void submit ( )
  {
    int age=14;
    if (age>=21)
    {
      s.o.p("Happy life...");
    }
    else
    {
      throw new MariageException("Invalid age");
    }
  }
  public static void main (String[] args)
  {
    try{
      submit();
    }
    catch (MarriageException e)
    {
      s.o.p(e.getMessage());
    }
  }
}
```

```
class MariageException
extends
Exception
{
  String msg="Invalid";
  MariageException (String msg)
  {
    this.msg = msg;
  }
  public String getMessage()
  {
    return msg;
  }
}
```

15/12/2022

-: COLLECTION :-

* Collection is an unified Architecture which consists of classes and interface.

* All the collection classes and interface belongs to java.util. package.

* In order to overcome drawback of array, we go for collection.

* In collection we can store hetrogeneous type and size is dynamic

* It increases its size by 50%.

* The default capacity is 10.

In collection we can have 3 sub Interface :-

1] LIST
2] QUEUE
3] SET

1] LIST :- List is a type of one of the Interface type of collection.

* LIST is an Interface whenever we want to store the elements upon index and allow duplicates then we should go for LIST type of collection.

Features of LIST :

1) size is dynamic
2) we can store heterogenous type of data.
3) it is indexed type of collection.
4) it allows duplicates.
5) it ~~allow~~ follows order of insertion
6) it allows null
7) since it is indexed type of collection we can fetch the element upon index randomly.
8) it is not auto sorted.

LIST has three sub class :

1) Array LIST
2) vector
3) LinkedLIST

1) Array LIST :

① features of Arraylist.

1) size is dynamic
2) we can hetrogenous type of data.
3) it is indexed type of collection
4) it allows duplicates.
5) it follows order of insertion

5] it allows null.

7] Since, it is indexed type of collection we can fetch the elements upon index randomly.

(8) it is not auto sorted

9] it is increased its size by 50%

10] it is not synchronized

11] Since, its not synchronized the performance is fast.

2] VECTOR :-

features of VECTOR :

1) size is dynamic

2) we can heterogeneous type of data

3) it is indexed type of collection

4) it allows duplicates

5) it follows order of insertion

6) it allows null

7) since, it's index type of collection we can fetch the elements upon index randomly.

8) it is not auto sorted

9) it increased it size by 100%

10) it is synchronized

11) since its synchronized the performance is slow.

7) LINKED LIST :

whenever we need proper order of insertion then we should go for Linked List. It has dual properties.

(a) i.e it inherits from both LIST AND QUEUE.

Features of LINKED LIST :

1) size is dynamic
2) we can store heterogeneous type of data.
3) it is index typed of collection.
4) it allows duplicates.
5) it follows order of insertion.
6) it allows null.
7) since, it is indexed type of collection we can fetch the elements upon index randomly.
8) It is not auto sorted
9) it increases its size by 50%

2] QUEUE :-

(1) Queue is an interface whenever we want to define general queue then we should go for queue type of collection.

→ There are have

1] LINKED LIST

2] PRIORITY QUEUE

1] LINKED LIST : JAVA provides an in LIST also work as Queue

2] PRIORITY QUEUE : It is a class which implements Queue interface which belongs to java.util package

Features of priority queue :

1] size is dynamic

2] we can store homogeneous type of data

3] It is not indexed type of collection

4] It does allows duplicates

5] It is partially sorted

6] it does not allow null

7] since it is not indexed type of collection we can't fetch it element, upon index random

3] **SET** :- set is an interface which helps to store the data not upon index and doesn't allow duplicates.

in set we have.

1] Hashset
2] Linked Hashset
3] Tree set.

features of set :-

1] size is dynamic
2] we can store heterogeneous type of data.
3] it is not indexed type of collection.
4] it does not allow duplicates
5] it allows ~~duplicates~~ null.
6] since, it is not indexed type of collection we can't fetch the elements upon index randomly.
7] it increases its size by 50%.

1] Hashset :- It is a class which belongs to java.util package.

features of Hashset :-

1] size is dynamic
2] we can store heterogenous type of data.
3] it is not indexed type of collection.
4] it doesn't allows duplicates
5] it allows null.
6] since it is not indexed type of collection we can't fetch the elements upon index randomly.
7] it increases its size by 50%.
8]. it doesn't follow order of insertion.

2]. LinkedHashset :- It is a class which belongs to java.util package.

features of Linked Hashset :-

1] size is dynamic.
2] we can store heterogenous type of data.
3] it is not indexed type of collection
4] it doesn't allows duplicates.
5] it allows null.
6] since it is not indexed type of collection we can't fetch the element upon index randomly.

7) it increases its size by 50%.
8) it follow order of insertion

2] Treeset :- it is an class which belongs to java.util package

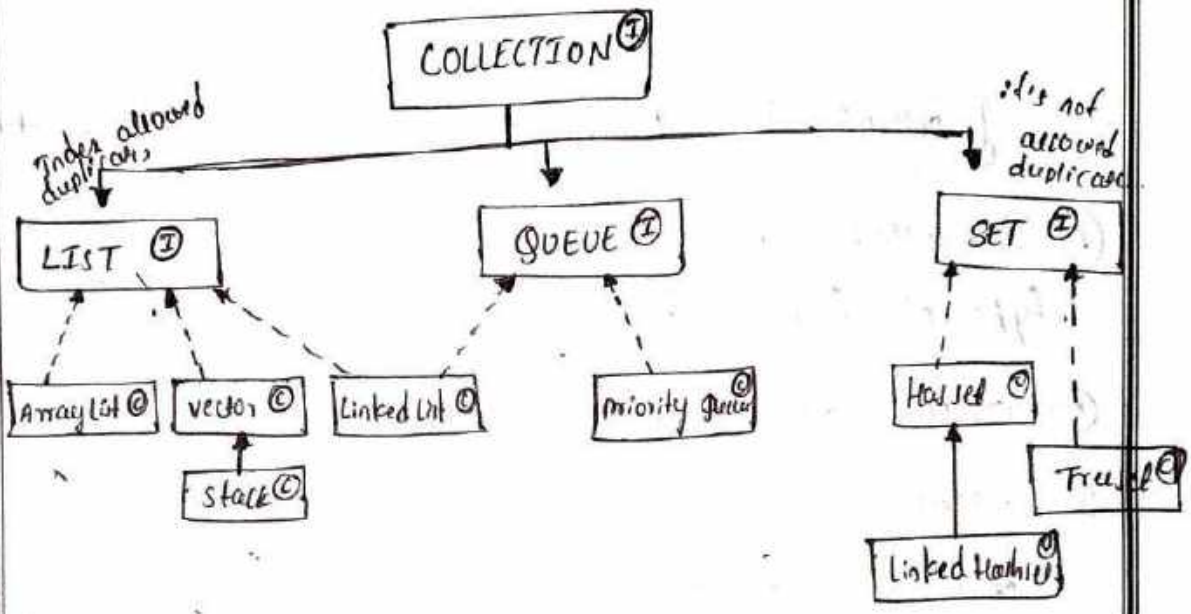
features of Treeset :-

1) size is dynamic
2) we can store homogenous type of data.
3) it ~~doesn't allows duplicates.~~
not indexed type of collection.
4) it doesn't allows duplicates.
5) it allows null.
6) since, it is not indexed type of collection we can't fetch the elements upon index randomly.
7) it is increases it size by 50%.
8) it is auto sorted.

COLLECTION :-

" collection is a unified Architecture which consist of Classes and Interfaces "

(*) In collection we have Three sub Interfaces



1) LIST
2) QUEUE
3) SET

1) LIST :- list is an Interface which allows duplicate and store the data upon index.

→ (*) In order to overcome the drawback of array we go for collection.

→ (*) that is In collection the size is dynamic & we can store Heterogeneous type of data.

(*) The Default capacity is 10.

Array

```
int arr = {10, 20, 30, 40};
```

- In Array we can store Homogenous type of data.
- The Array size is fixed, we can't change it.

collection default capacity is 10 $16 * \frac{3}{4} + 1 = 16$

- In collection is an Interface we can store Heterogenous type of data.
- collection size is dynamic. keep changing size.
- collection is a unified Architecture it's contains classes and interfaces.
- collection we have Three Sub Interfaces.

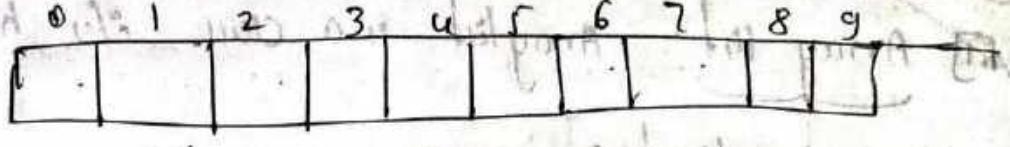
1] LIST
2] QUEUE
3] SET.

1] LIST :- List is an sub Interface of collection. it's allow duplicate and stores the data upon Index.

it's have sub classes :-

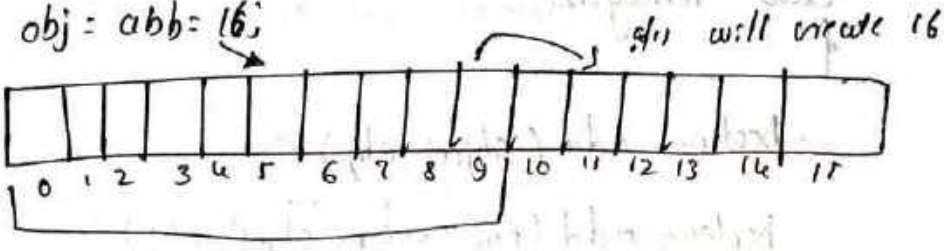
1.1] ArrayList
1.2] Vector
1.3] Linked list.
1.4] Stack.

The collection default size is 10.

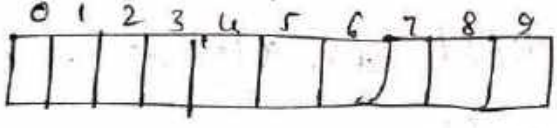


obj = arr = 10;

obj = abb = 16;



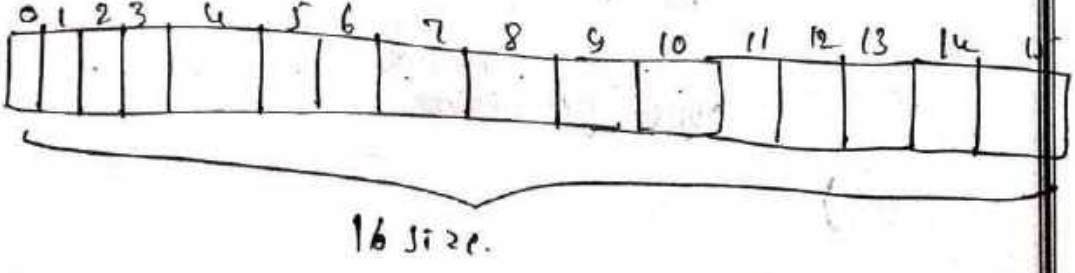
current capacity is = 10;



increase $= \overset{5}{\cancel{10}} * \frac{3}{\cancel{2}_1} + 1$

$= 5 * 3 + 1$

$= 16$



~~Question~~

Array list: Array list is a class. it's have some methods:

```
class Arraylist
{
    boolean add ( object obj)
    boolean add (int index, object obj)
    boolean addAll ( collection c)
    boolean addAll (int index, collection c)
    boolean remove ()
    void removeAll ()
    void retainAll ()
    boolean contains ()
    int size ()
    object get (int index)
}
```

(1) boolean add (object obj) :-

" boolea add (object obj) is method of Array list. It is used to add the elements in form of object."

(2) All the hetrogenous data type can add method with add the elements in form of object

Example :

```
class MainClass
{
    public static void main (String[] args)
    {
        Arraylist l1 = new Arraylist();
            l1.add(10);
            l1.add(20.56);
            l1.add('A');
            l1.add("Hello");
            l1.add(null);
            l1.add(10);
        S.O.P(l1);
    }
}
```

O/P

[10, 20.56, A "Hello, null 10]

② boolean add (int index, object obj): it's method of Arraylist. it's add method adding method with index Address form of object.

Ex:

```
~~class~~
package collectionTopic;
import java.util.Arraylist;
public class Sample1
{
  public static void main(String[] args)
  { ArrayList l1=new ArrayList();
    // l1.add(10);
    // l1.add(10.52);
    // l1.add('A');
    // l1.add("Hello");
    // l1.add(null);
    // l1.add(10);
    // s.o.p(l1);
```

← add method will add all Hetrogenous type of data.

```
    l1.add(2, true)
    s.o.p(l1);
```

← add (int index, object obj)

O/P

0	1	2	3 ...	n
0		true		

```
  }
}
```

② ~~boolean addAll A~~

② boolean addAll (collection c) : it's method of Arraylist it will add all elements to another object.

```
import java.util.Arraylist;

class Mainclass
{
    public static void main (String [] args)
    {
        Arraylist l1= new Arraylist ();
            l1. add(10);
            l1. add(20);
            l1. add(30);
            l1. add (40);

        Array l2=new Arraylist ();
            l2. add('A'); l2. add('A');
            l2. add ('B');
            l2. add ('C');
            l2 . add ('D');
        l1. addAll(l2);
    }
}
```

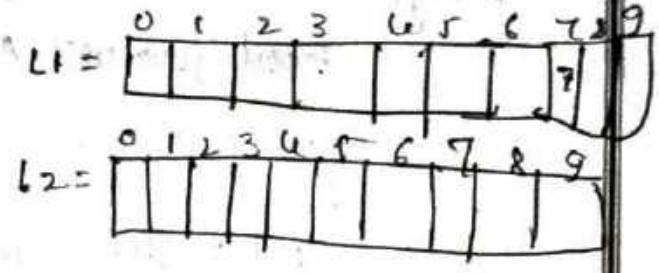
o/p

L1 = [10 , 20 , 30, 40 , A , B, C, D]

L2 = [A, B, C, D]

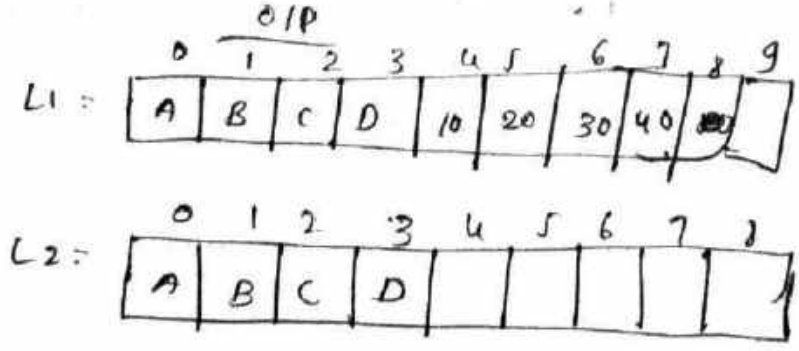
boolean addAll (int index, ~~object obj~~ Collection c) :-

All Arraylist method is used to add all elements with the index value.




```
import java.util.Arraylist
class MainClass
{
  public static void main (String[] args)
  {
    Arraylist l1 = new Arraylist();
        l1.add (10);
        l2.add (20);
        l1.add (30);
        l1.add (40);
    Arraylist l2 = new Arraylist();
        l2.add ('A');
        l2.add ('B');
        l2.add ('C');
        l2.add ('D');
    l1.addAll (0, l2);
  }
}
```

O/P



20/12/2022

(*) boolean remove (int index):

(*) remove method will removes the values.

(*) remove method will remove upon values and Index..

(*) its doesnt remove integer value where we have to remove upon index.

```
import java.util.Arraylist;

class Main class
{
    public static void main (String [] args)
    {
        Arraylist l1 = new Arraylist();
            l1.add("GOA");          //0
            l1.add("MANDVA");       //1
            l1.add("BIJAPURA");     //2
            l1.add("BANGALORE");    //3
            l1.add("DELHI");        //4
            S.O.P(l1);

        l1.remove("DELHI");   //remove values
        S.O.P (l1);

        l1.remove (2);        //remove Index address
        S.O.P (l1);
    }
}
```

O/P

[GOA MANDVA BIJAPURA BANGALORE, DELHI]

[GOA, MANDVA, BIJAPURA, BANGALORE]

[GOA, BIJAPURA, BANGALORE]

void removeAll () :

(*) removeAll will remove all the duplicates in L1 with respect to L2.

(*) removeAll will removes duplicate values.

void retainAll() :

(*) retainAll will retain all the duplicates in L1 with respect to L2.

(*) retainAll will display duplicates values.

```
class MainClass
{
  public static void main (String[] args).
  {
    ArrayList l1 = new ArrayList();
              l1.add (10);
              l1.add (20);
              l1.add (30);
              l1.add (40);
              l1.add (50);
    ArrayList l2=new ArrayList();
              ~~l2.add (40);~~
              ~~l2.add (50);~~
              l2.add (40);
              l2.add (50);
              l2.add (60);
              l2.add (70);
    l1.removeAll (l2);   //remove all duplicate values
  }
}
```

L1= | 10 | 20 | 30 | 40 | 50 |

duplicate value

L2= | 40 | 50 | 60 | 70 |

L1= | 10, | 20 | 30 |

in L2

O/P

L1 = [10, 20, 30, 40, 50]

update

L1= [10,20,30] L2= [40, 50, 60, 70]

~~retain~~
retainAll() :- it displays the duplicates values.

```
class MainClass
{
  public static void main (String[] args)
  {
    ArrayList l1=new ArrayList();
              l1.add(10);
              l1.add(20);
              l1.add(30);
              l1.add(40);
              l1.add(50);

    ArrayList l2=new ArrayList();
              l2.add(40);
              l2.add(50);
              l2.add(60);
              l2.add(70);
    S.O.P(l1);
    S.O.P(l2);
    ~~l1.retain(l2);~~
    l1.retainAll(l2);      // retainAll method
    S.O.P(l1);
    S.O.P(l2);
  }
}
```

O/P

L1 = [10, 20, 30, 40, 50]
L2 = [40, 50, 60, 70]

L1 = [40, 50]
L2 = [40, 50, 60, 70]

boolean contains () :

* contains () method will check the elements is present in the given collection, if it is present it will return true, use it will return false.

```
class Main class
{
  public static void main (String[] args)
  {
    ArrayList l1 = new ArrayList ();
              l1.add (10);
              l1.add(20);
              l1.add(30);
              l1.add (40);
              l1.add (50);
    boolean a = contain (10);
        S.O.P (a);

  }
}
```

O/P

True

Object get (int index) :-

get method will get elements upon Index.

```
class Mainclass
{
  public static void main (String[] args)
  {
    ArrayList l1=new ArrayList();
              l1.add (10); //0
              l1.add (20); //1
              l1.add (30); //2
              l1.add (40); //3
              l1.add (50); //4.

              l1.get (10);   //getting to element
              s.o.p (l1)

    for (int i=0; i< l1.get(); i++)
    {
      Object o1 = l1.get(i);
      s.o.p (o1);
    }

  }
}
```

O/P

10
20
30
40
50

10

int size () function :

(k) size function will calculate how many elements are present in the given collection.

```
class Main class
{
  public static void main (String [] args)
  {
    ArrayList al = new ArrayList ();
              al.add (10); //0
              al.add (20); //1
              al.add (30); //2
              al.add (40); //3
              al.add (50); //4

    for ( int i = 0; i < al.size(); i++)
    {
      Object o = al.get();
        s.o.p (o);
    }
  }
}
```

O/P

```
10
20
30
40
50
```

21/12/2022

2] **VECTOR** :- vector is a inbuilt class its have same method, is there in what we learnt in Array list.

(*) CONSTRUCTOR OVERLOADING :-

(*) "Developing multiple constructor within the class but variation in the Arguments List is called as constructor overloading."

(*) variation in the Arguments list means, variation in the data type, variation in the length of the Argument, variation in the order of occurence of the Arguments.

(*) In vector class constructor is overloaded

1] vector ( ) :- it will initalize with default capacity 10.

2] vector (int initial_cap) :- it is used to initilize, as per the user required capacity.

3] vector (collection c) :- its helps to copy from one collection object to another collection object.

```
class sample
    vector ( )
    {
        int default_cap = 10;
    }
    vector (int initial_cap)
    {
    }
    vector (collection c)
    {
    }
}
```

VECTOR :

* vector is a class which belongs to java.util.~~lang~~ package.

Arraylist	vector
* it increases size by 50%.	* it increase size by 100%
* Arraylist is not synchronized	* vector is synchronized
* Arraylist is fast	* vector is slow.

```
package vector;
import java.util.vector;
import java.util.Arraylist;

class Main class
{
  public static void main (String[] args)
  {
    vector l1 = vector(3);
            l1.add(10); //0
            l1.add(10.5); //1
            l1.add('A'); //2
            l1.add("Hi"); //3

    S.O.P ("capacity:" + l1.capacity());
    S.O.P ("size:" + l1.size());

    Arraylist l2 = new Arraylist(l1); //copying o/p from l1 elements
    S.O.P (l2);
```

size of element we should store

we can give 3 size it will increase the double size 6.

O/P

capacity 6

size: [10 10.5 A Hi]

}

}

For-each loop :-

(R) for-each is looping, it's process the elements one by from the given reference variable. is not given index number.

syntax :-

```
for ( datatype  variable name :  ref variable )
{
    ==  //variable name
}
```

(1)
```
int[] arr = {10, 20, 30, 40};

for ( int  a1 : arr )
{
    s.o.p (a1);
}
```

o/p

10, 20, 30, 40, 50

(2)
```
boolean arr = {10.2, 55.6, 10.5};

for ( boolean  a2 : arr )
{
    s.o.p (a2);
}
```

10.2, 55.6, 10.5

③ char arr = {'A', 'B', 'C', 'D'};

```
for ( char a3 : arr)
{
  S.O.P (a3);
}
```

o/p

A , B , C , D

```
class main class
{
  public static void main (String[] args)
  {
    vector l1 = vector (5);
        l1.add (10);
        l1.add (10.5);
        l1.add ('A');
        l1.add ("Hi");
        l1.add (10);
    for ( object a1 : l1)        // for_each loop
    {
      S.O.P (a1);
    }

/*  for ( int i=0; i<l.size; i++)
      {  Object o1 = l1.get(i);        // for loop
      }    S.O.P(o1);   */

  }
}
```

Vector :-

(1) vector is a class it's belongs to java.util package.

(2) vector is 100% increase size.

(3) vector class, there is a inbuilt methods. what we learn previous class Arraylist. methods same methods will be apply in vector.

```
import java.util.Vector;
class Mainclass
{
  public static void main(String[] args)
  {
    boolean add (object obj);
    boolean add (int index, object obj);
    boolean addAll (collection c);
    boolean addAll (int index, collection c);
    void remove ();
    void removeAll ();
    void retainAll ();
    void contains ();
    object get (int index);
    int size ();
  }
}
```

① ~~object~~ boolean add (object obj) :- it will all Hetrogenous type of elements.

```
import java.util.vector;
class Main class
{
  public static void main (String[] args)
  {
    vector v1 = new vector(5);
        v1.add (10);
        v1.add (10.5);
        v1.add ('A');
        v1.add ("HELLO");
        v1.add (10);
  }     S.O.P (v1);
}
```

O/P

	0	1	2	3	4
v1 =	10	10.5	A	HELLO	10

② boolean add (int index, object obj) :- it will add the elements based on index address.

```
import java.util.vector;
class Main class
{
  public static void main (String[] args)
  {
    vector v1 = new vector (5);
        v1.add (10);    //0
        v1.add (10.3);  //1
        v1.add ('H');   //2
        v1.add ("JAVA");//3
        v1.add (10.3);  //4
      v1.add (3, 'I');
    S.O.P (v1);
  }
}
```

O/P

	0	1	2	3	4	5
v1 =	10	10.3	H	I	JAVA	10.3

(3) boolean addAll (Collection c):- it will display ~~matched elements~~ ~~& duplicate values will remove the elements from one collection to another collection.~~ all element in one collection

```
import java.util.vector;
import java.util.ArrayList;

class Main class
{ public static void main (String[] args)
  { vector v1=new vector(3);
        v1.add(10);
        v1.add(20);
        v1.add(30);

    ArrayList l1=new ArrayList();
    ~~s.o.p (l1);~~ l1.add(40);
                    l1.add(20);
                    l1.add(30);

        l1.addAll(v1);
        s.o.p(l1);
  }
}
```

l1 = [10, 20, 30, 40, 20, 30]
[40, 20, 30]

v1 = | 10 | 20 | 30 |

l1 = | 40 | 20 | 30 |

l1 = | 10 |

v1 = | 40 | 20 | 30 |

(4) boolean addAll (int index, collection c):- ~~it will display duplicate values~~

```
import java.util.vector;
import java.util.ArrayList;

class Main class
{ public static void main (String[] args)
  { vector v1=new vector(9);
        v1.add(10);
        v1.add(10.5);
        v1.add("HE");

    ArrayList l1=new ArrayList();
        l1.add(30);
        l1.add(40);
        l1.add("JAVA");

    l1.addAll(v1)
    s.o.p(l1);
  }
}
```

v1 = | 10 | 10.5 | HE | Hi | 30 | 40 | JAVA |

(5) void remove (): remove method will remove the values (elements based one values & Index address.

```
import java.util.Vector;

class MainClass
{
  public static void main (String [] args)
  {
    Vector v1 = new Vector (4);
        v1.add (10); //0
        v1.add (20); //1
        v1.add (30); //2
        v1.add (40); //3
    v1.remove (10);  ---> values will remove
    v1.remove (1,20); --> Index address / that value will remove.
    s.o.p (v1);
  }
}
```

O/P

v1 = [30 | 40]

(6) void removeAll (): it will removeAll the one collection to another collection matched duplicates values.

```
import java.util.Vector;
import java.util.Arraylist;

class MainClass
{
  public static void main (String [] args)
  {
    Vector v1 = new Vector (3);
        v1.add (10);
        v1.add (20);
        v1.add (30);
    Arraylist l1 = new Arraylist();
        l1.add (40);
        l1.add (20);
        l1.add (30);
    l1.add (v1);
    s.o.p (l1);
  }
}
```

v1 = [10 | 20 | 30] (0, 1, 2)

l1 = [40 | 20 | 30] (0, 1, 2)

O/P

[10]

[40, 20, 30]

(7) void retainAll (); it's display duplicate values.

```
import java.util.Vector;
import java.util.ArrayList;
class MainClass:
{ public static void main (String[] args)
  {
    Vector v1=new Vector(3);
          v1.add(10);
          v1.add(20);
          v1.add(30);
    ArrayList l1=new ArrayList();
          l1.add(40);
          l1.add(20);
          l1.add(30);
    l1.retainAll(v1);
    S.o.p(l1);
  }
}
```

v1 = | 10 | 20 | 30 | (0 1 2)

l1 = | 10 | 20 | 30 |

o/p

| 20 | 30 |

l1 | 40, | 20 | 30 |

(8) void contains (); it will display what elements present in the collection.

```
import java.util.Vector;
class MainClass
{ public static void main(String[] args)
  {
    Vector v1=new Vector(5);
          v1.add(10);
          v1.add(10.5);
          v1.add('H');
          v1.add("HELLO");
          v1.add(10);
    boolean = contains(10);
    S.o.p(2);
  }
}
```

o/p

True

(8) object get(int index) :- It will get the elements & display it.

```
import java.util.Vector;
class Mainclass
{
  public static void main (String[] args)
  {
     Vector v1=new Vector (4);
            v1.add (10);
            v1.add (20);
            v1.add (30);
            v1.add (40);
       v1.get (10);
      S.o.p (v1);
    for( int i=0; i< v1.get(); i++)
     {   object ol= v1.get(i);
         S.o.p (ol);
     }
  }
}
```

O/P

10

(10) int size() :- It will check the range

```
import java.util.vector;
class Mainclass
{ public static void main ( String[] args)
  {  vector v1=new vector (5);
            v1.add(10);
            v1.add (20);
            v1.add (30);
            v1.add (40);
            v1.add (50);
    for( int i=0; i< v1.~~get()~~ size(); i++)
     {  object ol= v1.get();
        S.o.p (ol);
  } }
}
```

O/P

10
20
30
40
50 //

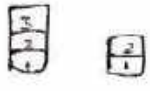
22/12/2022

2 QUEUE :- Queue is a Interface of collection, It have classes LinkedList & Priority queue

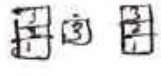
1) Priority Queue :- priority queue is ~~call~~ class which implements ~~que~~ queue Interface it's will not allow Index.

(2) to fetch the elements poll() and peek().

1] poll method :- it will fetch the top most elements of and remove the elements of queue. its size will reduced by one.



2] peek() method :- it is a non-static method of priority of queue class. it will fetch the top most element and doesnot remove the element from queue.



(*) "priority queue is auto sorted"

(*) it will display in sequencial order.

Priority queue :-

(1) poll () :- it will fetch top most element & remove element from queue.

(2) peek () :- it will fetch top most element & doesn't remove element from queue.

poll () :-

```
import java.util.PriorityQueue;
class Main class
{
    public static void main (String[] args)
    {
        PriorityQueue l1 = new PriorityQueue();
                l1.add(20);
                l1.add(10);
                l1.add(15);
                l1.add(5);

        S.O.P (l1);
        S.O.P (l1.poll()); // poll method
        S.O.P(l1);
        S.O.P (l1.poll()); // poll method

    }
}
```

O/P

```
[5  10  15  20]
5
[10  15  20]
10
[15  20]
```

Peek() :-

```
import java.util.priorityQueue;

class MainClass
{
    public static void main (String[] args)
    {
        priorityQueue l1 = new priorityQueue();
            l1.add (20);
            l1.add (10);
            l1.add (15);
            l1.add (5);

        s.o.p(l1);
        s.o.p(l1.peek());    // peek method
        s.o.p(l1);
        s.o.p(l1.peek());    // peek method.
        s.o.p(l1);

    }
}
```

O/P

```
l1 = [5   10   15   20]
     5
     [5   10   15   20]
     10
     [5   10   15   20]
```

for each loop :-

```
import java.util.PriorityQueue;

class MainClass
{
    public static void main(String[] args)
    {
        PriorityQueue l1=new PriorityQueue();
                l1.add(20);
                l1.add(10);
                l1.ad(15);
                l1.ad(5);

        for (Object o1 : l1)
        {
            S.O.P(o1);
        }

    }
}
```

O/P

5
10
15
20

1) LinkeList :

(*) LinkeList is a class, which has Dual property that is which implement LIST & queue Interface.

(*) Whenever we want to have proper order of insertion then we should go for Linked List.

(*) LinkeList have ~~also~~ ~~some~~ methods poll(), peek()

```
import java.util.LinkedList;

class MainClass
{
    public static void main(String[] args)
    {
        LinkedList ll=new LinkedList();
                ll.add(10); //0
                ll.add('A'); //1
                ll.add("hello"); //2
                ll.add(20.56); //3
                ll.add(10); //4
            S.O.P(ll);
            S.O.P(ll.get(1));
            S.O.P(ll);
            S.O.P(ll.poll());
            S.O.P("all-poll-->"+ll);
        }   S.O.P(ll.peek());
            S.O.P("all---peek-->"+ll);
    }
```

~~Note~~: Demonstrate All the method of Linked List & priorityQueue.

O/P

[10 . A Hello 20.56 10]

A

[10, A Hello 20.56 10]

10

a4 ... Poll ---> [A hello 20.56 10]

A

a4 ... peek ---> [A hello 20.56 10]

3] SET :-

23/12/2022

(*) SET is an Interface which belongs to java.util. package

(*) In SET we have

1] Hash set 2] Linked Hashset 3] Treeset.

1] Hashset :-

(*) SET doesnt allowed duplicates, & not indexed.

1] Hash set :- it will not follow order of insertion.

2] Linked Hashset :- it will follow order of insertion.

3] Treeset :- it completely sorted.

1] Hash set :-

```
import java.util.Hashset;
class Sample
{
  public static void main (String[] args)
  {
    Hashset ll= new Hashset ();
        ll.add (10);
        ll.add (20.56);
        ll.add ("hello");
        ll.add ('A');
        ll.add (10);        // S.O.P (ll);
    for( Object ob : ll)
        S.O.P (ob);
  }
}
```

O/P

A 20.56 hello 10

A
20.56
10
hello

2] Linked Hashset :-

```
import java.util. ~~Hashset~~ LinkedHashset;

class sample
{
  public static void main (String[] args)
  {
    LinkedHashset l1 = new LinkedHashset ();
                l1.add (10);
                l1.add (20.56);
                l1.add ('A');
                l1.add ("Hello");
                l1.add (10);
        // s.o.p (l1);
      for ( Object o1 : l1)
        {
          s.o.p ( o1);
        }
  }
}
```

O/P

10	20.56	A	Hello

O/P

10
20.56
A
Hello

3] Treeset :-

```
import java.util.Treeset;
class sample
{
  public static void main (String[] args)
  {
    Treeset t1 = new Treeset();
                t1.add (50);
                t1.add (20.56);
                t1.add (30);
                t1.add (40);
                t1.add (10);
        //s.o.p (t1);
    for (Object o1 : t1)
        {
          s.o.p (o1);
        }
}
```

O/P

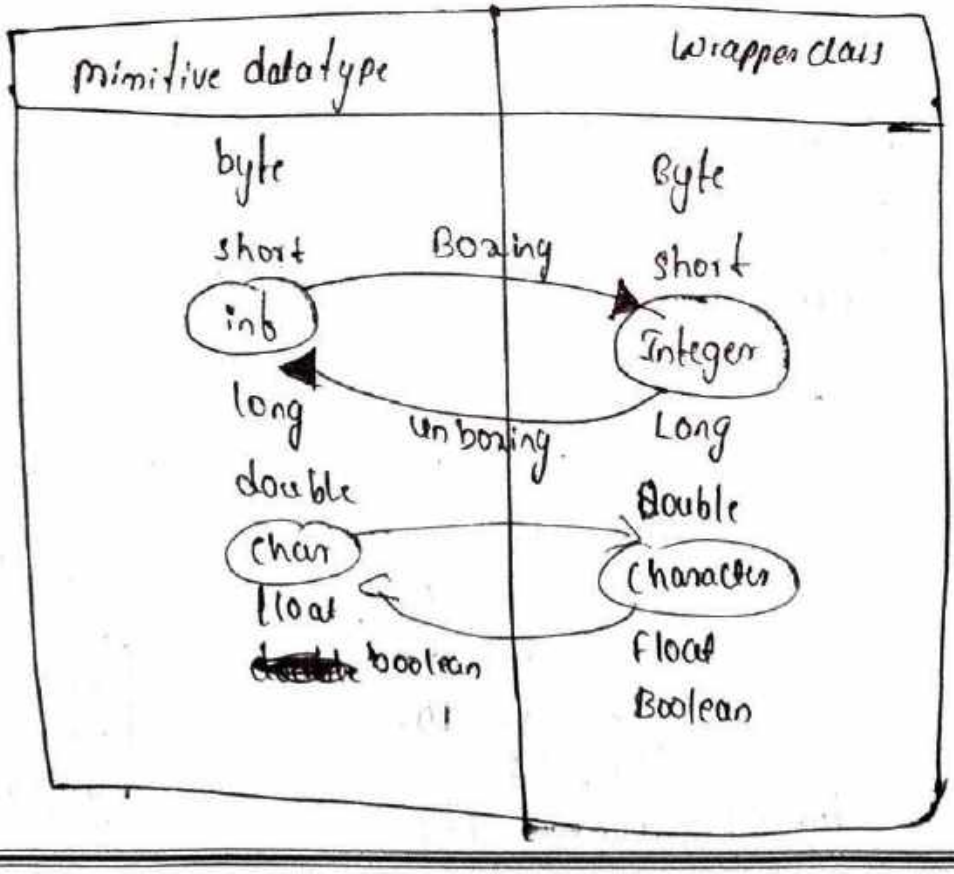
10	20.56	30	~~Hello~~ 40	d

O/P

10
20.56
30
~~Hello~~ 40
50

Wrapper class :-

- wrapper class are the inbuilt classes which belongs to java.lang package.
- for each & every primitive data type there is a corresponding class called wrapper class.
- converting from primitive data type to ~~any of its wrapper class~~ corresponding wrapper class object is called a Boxing.
- Converting back from wrapper class object to corresponding primitive type is called as unboxing.



② How the Elements are stored in the form of object in collection :-

```
ArrayList l1=new ArrayList();
l1.add(10);->object o1=new Integer(10);
l1.add(20.5);->object o1=new Double(20.5);
l1.add('A');  Object o1=new Character('A')
l1.add(true); Object o1=new Boolean(true);
for(object a1:l1)
{ s.o.p(a1);
}
```

Collections :-

(*) collections is class which belongs to java.util. package.

(*) it has a sort method. which will sort the elements in ascending order.

(*) example:-

```
import java.util.ArrayList;
import java.util.Collections;

class Sample
{
    public static void main (String[] args)
    {
        ArrayList l1 = new ArrayList ();
                l1.add (10);
                l1.add (50);
                l1.add (5);
                l1.add (40);
                l1.add (10);
        S.o.p (l1);
        Collections.sort (l1);
        S.o.p (l1);
    }
}
```

o/p

```
[10, 50, 5, 40, 10]
[5, 10, 10, 40, 50]
```

WAP to add Array Elements to Arraylist.

```
import java.util.Arraylist;
class Sample
{
public void main
int [] arr = { 10, 20, 30, 40, 50};
                0   1   2   3   4

Arraylist ll=new Arraylist();
        ll.add (arr[0]); //10
        ll.add (arr[1]); //20
        ll.add (arr[2]); //30
        ll.add (arr[3]); //40
        ll.add (arr[4]); //50
        s.o.p (ll); // print the elements.

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

array elements add to ll.

O/P

ll

10
20
30
40
50

Note: in array we can store Hetrogenous type of data.

Assignment

write 8+8 program to add the elements from array to collection or from collection to Array.

WAP to add collection elements to Array.

```
import java.util.ArrayList;
class sample
{
    public static void main (String[] args)
    {
        ArrayList l1 = new ArrayList();
                l1.add(10); //0
                l1.add('A'); //1
                l1.add(20.56); //2
                l1.add(true); //3

        object[] arr = new object[l1.size()]; //4
```

0	1	2	3
10	A	20.56	true

```
        arr[0] = l1.get(10);
        arr[1] = l1.get('A');
        arr[2] = l1.get(20.56);
        arr[3] = l1.get(true);

        for( int i=0; i<l1.size(); i++)        //storing data
        {                                       from collection
            arr[i] = l1.get[i];                 to array.
        }

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

-: MAPS :-

24/12/2022

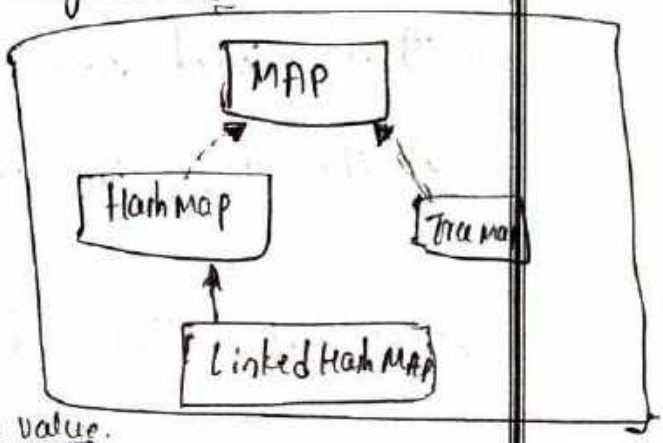
(*) MAPS is an interface which belongs to java.util package.

(*) In map we have three sub classes

(*) Hash map (*) LinkedHashMap (*) TreeMAP

```
class Map
{
  Put();
  Put All();
  Entry entryset()
  { get value();
    get key();
```

(*) Whenever we want store upon key & values then we should go for MAP.



(*) Features of MAP :-

1) size is dynamic.
2) it is by default generic.
3) it's stores data upon key & value.
4) it's doesn't allow duplicate keys
5) it's allows duplicate ~~duplicate~~ values

Keys	value
name	Phone
Nga	636[illegible]
Naru	9772165[illegible]

Hash map :-

(*) Hash map is a class which implements ~~Hash~~ MAP Interface.

(*) features of HashMAP :-

1) size is dynamic
2) it is by default generic.
3) it stores data upon key & value.
4) it's doesn't allow duplicate keys.
5) it's allows duplicate values.
6) it will not follow of order insertion.

2] Linked Hash Map :-

(*) Linked Hash map is a class which extends Hash Map

Features of Linked Hash Map :-

1] size is dynamic.

2] it is by default generic.

3] it's stores data upon key a value.

4] it's doesn't allow duplicates key.

5] it's allows duplicates values.

6] it will follow order of insertion.

3] Tree Map :-

(*) Tree Map is a class which implements MAP Interface.

Features of Tree Map :-

1] size is dynamic.

2] it is by default generic.

3] it's stores data upon key a value.

4] it's doesn't allow duplicates key.

5] it's allows duplicates values.

6] it is auto sorted upon keys.

GENERIC :- < >

① Generic it is used to store homogeneous type of data.

② by using Angular ~~bracket~~ symbol < >.

③ We can achive generic where it can store homogenous type of data.

④ we can specify any ~~Angular~~ class name with Angular ~~bracket~~ symbol.

Program:

```
import java.util.Arraylist;
class sample
{
    public static void main (String[] args)
    {
        Arraylist<Double> al = new ArrayList<Double>();
                al.add(10.23);
                al.add(20.56);
                al.add(5.23);
                al.add(9.56);
                al.add(10.23);

        S.O.P(al);
```

O/P

[10.23, 20.56, 5.23, 9.56, 10.2]

}

J Hashmap :-

```
package collectionstopic;
import java.util.HashMap;
import java.util.Map.Entry;

class Demo2
{
public static void main (String [] . arg)
{
HashMap<String, Integer>  ll=new  HashMap<String, Integer>();
                                                  key
        ll.put("raju", 823);
                                                  value.
        ll.put("sam", 113);
        ll.put ("mom", 823);
        ll.put ("sam", 112);
        ll.put ("dad", 154);
for (Entry  al: ll.entrySet())
{
  S.O.P (al.getKey()+" " al.getVal());
}

}
}
```

O/P

raju	823
~~mom~~ sam	112
mom	823
dad	154

2) LinkedHashMap :

```
import java.util.LinkedHashMap;
import java.util.Map.Entry;

Class Demo2
{
  public static void main (String[] arg)
  {
    LinkedHashMap <String,Integer> ll=new LinkedHashMap<String,Integer>();
        ll.put ("raju", 823);
        ll.put ("sam", 513);
        ll.put ("mom", 823);
        ll.put ("dad", 154);
        ll.put ("sam", 112);
    for (Entry al: ll.entrySet())
    {
      S.O.P (al.getKey()+"  "al.getValue());
    }
  }
}
```

O/P

raju	823
sam	112
mom	823
dad	154

3) Tree Map :-

```
import java.util.TreeMap;
import java.util.Map.Entry;

class Demo3
{
  public static void main (String[] args)
  {

    TreeMap<String,Integer> t1=new TreeMap<String,Integer>();
            t1.put("raju", 823);
            t1.put("sam", 513);
            t1.put("mom", 823);
            t1.put("dad", 154);
            t1.put("sam", 112);
    for (Entry e1: entrySet())
    {
      S.o.p(e1.getKey()+" " e1.getValue());
    }

  }
}
```

O/P

dad	154
mom	823
raju	823
sam	112

:- FILE HANDLING :-

(*) File is a class which belongs to java.io. package.

(*) File is nothing but similar kind of data, and performing operations on file. Like.

1] Create, 2] write, 3] Read, 4] delete, 5] update the Data.

performing this operation on a file is called as file handling.

```
import java.io.file
class file
{
  public static void main(String[] args)
  {
    boolean mkdirs()
    boolean exist()
    boolean delete()
    boolean create new folder().
  }
}
```

Program :- File :

```
import java.io.File
class Demo5
{
  public static void main (String[] args)
  {
    File f1 = new File ("c:\\User\\Jip\\Desktop\\folder1");

    if (f1.mkdir())  ---------> // Creating directory.
    {
      s.o.p ("folder is created...");
    }
    else {
      s.o.p ("folder is not created");
    }

    if (f1.exists())  ---------> // it's checks folder is there/not
    {
      s.o.p ("folder is exists");
    }
    else
    {
      s.o.p ("folder is not exists");
    }

    if (f1.delete())  ---------> // delete folder
    {
      s.o.p ("folder is deleted...");
    }
    else {
      s.o.p ("folder is not deleted");
    }

  }
}
```

(1) step1 :- creating file

27/12/2022

WAP to create text file.

```
import java.util.io.File;
class File
{
    public static void main (String[] args)
    {
        File f1 = new File ("c:\\user\\Desktop\\test mg.txt");
        if (f1.~~mkdirs~~ CreateNewFile())
        {
            s.o.p ("file is created...");
        }
        else
        {
            s.o.p ("file is not created....");
        }
    }
}
```

.xls
.pdf.
.txt

(2) Create New file() :- creating new file we will create different type of files like text, pdf, xls, etc. with the extensions

(1) simple.text
(2) simple.pdf
(3) simple.xls
(4) simple.word

② Step2: FileWriter() & File reader() :-

- File writer and file Reader are inbuilt classes which belongs to java.io package.
- In file writer class we have writer method which helps to write the data into file.
- In file Reader class we have read method which helps to read the data into file.

WAP to write the data into file :-

```
import java.io.File;
import java.io.IOException;
class Demo
{
  public static void main (String[] args) throws IOException
  {
    File f1 = new File ("C:\Users\Desktop\testmsg.text");
    FileWriter fw = new FileWriter (f1, true);  //append
    fw.write ("Hello", true);  //Append
    fw.flush();  // flush method is inserting data into file
    S.O.P ("data is written");
```

keyword previous data also visible.

O/P.

employee is working in
Hello

```
  }
}
```

[redacted]

File reader (): :

WAP to read the data from the file.

```
import java.io.File;
Class Demo
{
  Public static void main (String[] args)
  {
    File f1 = new File ("C:\\Users\\Desktop\\textmsg.text");
    File Reader fr = new FileReader (f1);
    char[] arr = new char[(int) f1.length()]; // to store memory
        fr.read(arr);
    String s1 = new String (arr);
    S.O.P (s1);
  }
}
```

buffer writer () and buffer Reader () :-

(R) buffer writer() and buffer reader() are the inbuilt classes which helps to improve the performance of writing and reading the data.

(I) buff-writer () and buffer reader () it is used to improve the performance of writing & reading file.

(V) Buffer writer () :

```
import java.io.Exception;
import java.io.file
import java.io.BufferWriter; import java.io.writer;
Class Demo
{
  Public static void main (String [] args) throws Exception
  {
    File f1 = new File ("c:\\User\\Desktop\\testmsg.txt");
    FileWrite fw = new FileWriter (f1);
    BufferedWriter bw = new BufferedWriter(fw);
        bw.write ("hello Java");
        bw.write new line ();
        bw.write ("cool Java");
        bw.newLine();
        bw.write ("cool Java");
        bw.flush();
    S.O.P("data is written");
  }
}
```

O/P

```
hello Java
cool Java
cool Java
```

(2) Buffer reader();:-

```
import java.io.Exception;
import java.io.File;
import java.io.reader; import java.io.BufferReader;
Class Demo
{
    public static void main(String[] args) throws Exception.
    {
        File fi = new File("c:\\User\\ Desktop\\ test msg.text")

        FileReader fr = new FileReader(fi);

        BufferedReader br = new BufferedReader(fr);

            String s1 = br.readLine();

            while( s1 != null)
            {
                s.o.p (s1);

                s1 = br.readLine();
            }

    }
}
```

~~Difference b/w throw~~

Dynamic Inputs :-

Take input from user Input Stream Reader (System.in);

```
import java.io.Exception;
import java.io.Bufferreader;
import java.io.InputStreamReader;
class Demo
{
    public static void main (String[] args) throw IOException
    {
        // InputStreamReader isr = new InputStreamReader(System.in);
        Bufferreader br = new Bufferreader (new InputStreamReader
                                            (System.in));
        S.O.P("enter data");
        String s1 = br.readline();
        S.O.P (s1);

    }
}
```

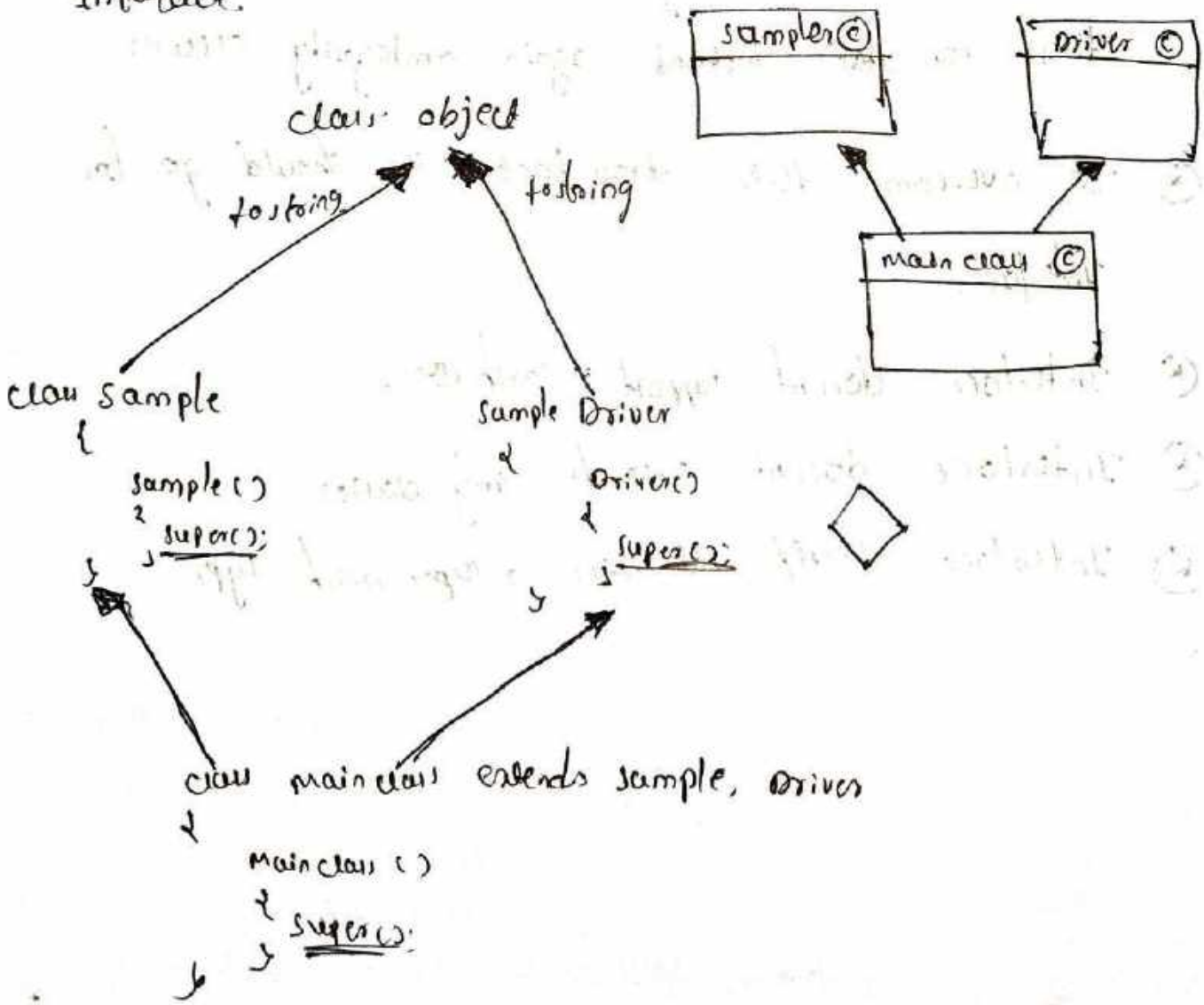
30/12/2022

Diamond problem | Ambiguity :-

Multiple Inheritance :-

" A sub class inheriting properties from one or more super class is called as multiple Inheritance."

- Multiple Inheritance we can achive through Interface.
- In class we can't achive the multiple Inheritance.
- Because, a sub class extends one or more super classes. which one super class properties i will inherited. the sub class get confused. so that's why we go for Interface.



- Each and every class will be having default constructor.
- Inside, the default constructor super() calling stmt, will be present.
- The Job of super() calling stmt is to call the parent class constructor.
- In case of multiple Inheritance sub class super() calling stmt will get confused and ambiguity occures.
- Each and every class instance object class. in object class we have many non-static methods. because of those non-static method again ambiguity occures.
- To, overcome this drawbacks we should go for Interface.
- Interface doesn't support constructor.
- Interface doesn't extends any classes.
- Interface itself act likes a super most type.

finalize():

- finalize() is an non-static and non final method of Object class.
- whenever object got a reference to null, gc() of system class will call the finalize method.
- finalize() method will do garabage collection.
- In Java grabage collection will be done implicitly.

- can we call the global variables type of object call the that method in a main method. yes.

because that is a non-static method so then we should create the object and make as a static objects. then calls the class name.

```
class mainclass {
{
 static public void finalize()
  {
    s.o.p("done");
  }
static mainclass1 mainclass1 = new mainclass1 ();
static mainclass2 mainclass2 = new mainclass ();
public static void main (String [] args)
{
        mainclass1 = null;
        mainclass = null;
}       system.gc();   // garbage collection
}
```

final	finally	finalize
⊛ final is a keyword	⊛ finally is a block	⊛ finalize is a non-static method.
⊛ Any members which is declared as a final. we can't modifying	⊛ Even if the exception is address, not address finally block will get executed.	⊛ finalize method helps to do garbage collection

Constructor Chaining :-

" calling a constructor inside another constructor is called as constructor chaining."

* To achive constructor chaining we have two stmts
  1) this() calling 2) super() calling.

1) this() call :-

* To achive constructor chaining within the class we have to use this() calling stmts

rules :-

* this() call stmt should be always first stmt. inside a constructor.

* we can't have more than one this() calling stmt.

* this() calling stmt can be use only inside a constructor

```
class sample
{
  public sample()
  {
    s.o.p("Hello");
  }

  sample(int no)
  {  this();
     s.o.p("Hii");
  }

  sample(int no, string b)
  {
    this(10);
    s.o.p("Bye");
  }

  public static void main(String[] a)
  {
    sample s = new sample();
           (10, "Sri");
  }
}
```

o/p

Hello
to
bye

2) super() call :-

(*) "To achive constructor chaining b/w the classes we have to use super() calling stmt."

(*) The Job of super calling stmt is to call the immediate parent class constructor.

rules :-

(*) super() calling stmt should be always first stmt inside a constructor.

(*) we can't write more than one super calling stmt.

(*) super calling stmt can't be used outside the constructor.

Note :-

(*) we can't use this() calling & super() calling in a single constructor.

```
class sample
{
    sample()
    {
    }
}

class Demo extends sample
{
    Demo( int no , string s)
    {
        super();
    }
}
```

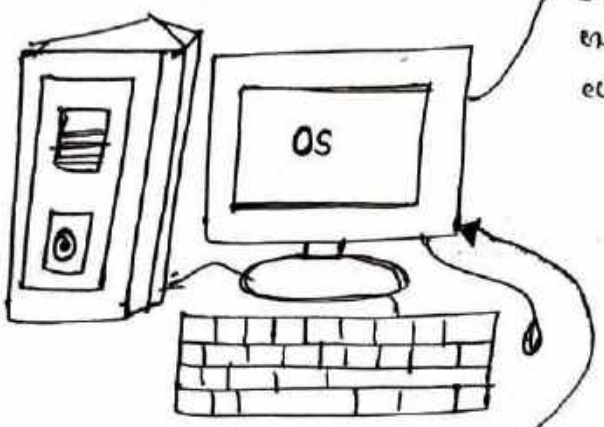
```
class MainClass
{
    ~~sub~~ P.S.V.m()
    {
        Demo d1= new
            Demo ()
    }
```

example :

```
class Test
{
    Test ()
    {
     s.o.p ("Hello");
    }


}
class Demo extends Test
{
    Demo ( ~~int no~~ , string b)
     super ();
     s.o.p ("Hi");

}
class. Driver extends Demo
{
    Driver (int no , string name)
    {
      super ();
      s.o.p ("Bye");
    }
~~class~~
public static void main (string[] args)
{
  Driver d= new Driver (10 ,"Naga");
```

28/12/2022

THREAD'S :-



Memory

	CPU	Time
word	1.8.	1s
excel	2.0	2s
ellipse	0.0	3s

applications runs in OS

word, excel, ppt, ellipse.

" Threads is an Execution Instance, which owns, it owns CPU time and memory " is called Thread.

① Thread is a class which belongs to java.lang package.

Slep() :-

① It is a static method of thread class, which will pause its execution for few mili seconds.

```
class Demo
{
    public static void main (String[] args)
    {
        for (int i=0; i<=10;, i++)
        {
            System.out.println (i);
            Thread.sleep(1500); // slep 1 and half seconds pause & start.
        }
    }
}
```

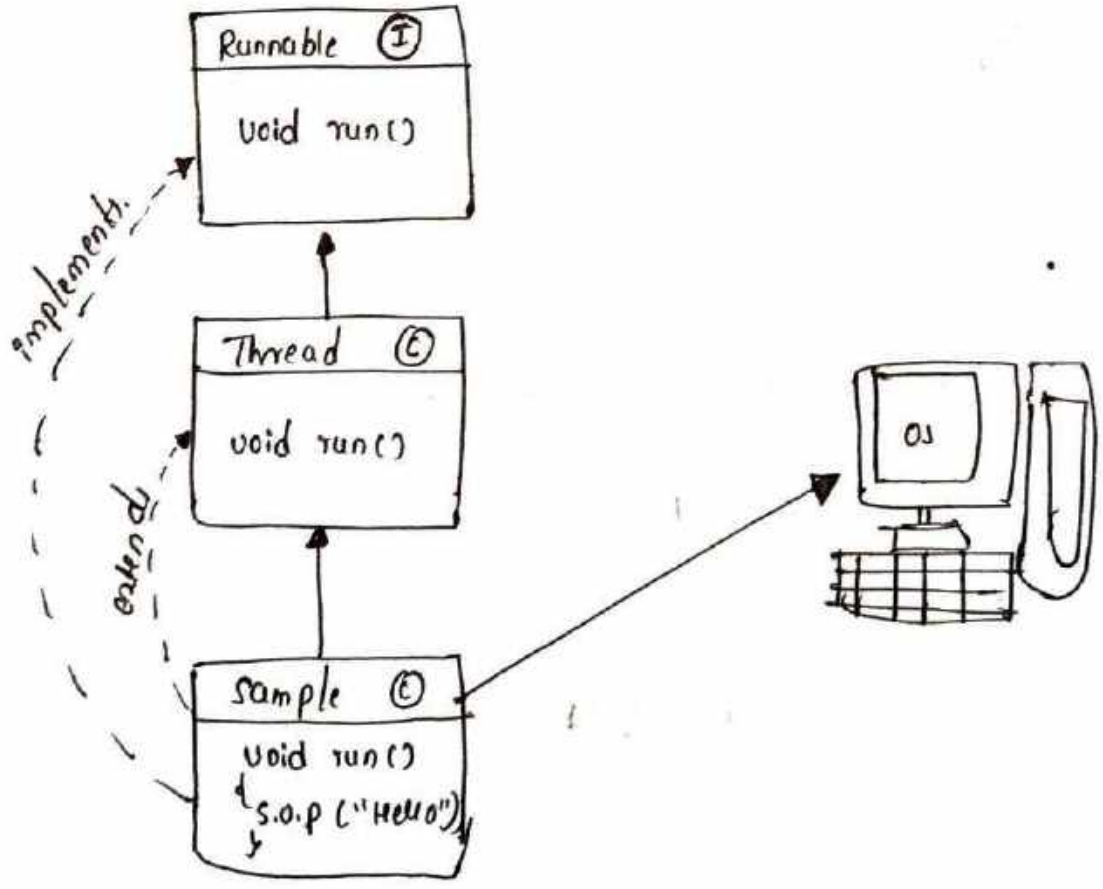
O/P

1
2
3
.
.
.

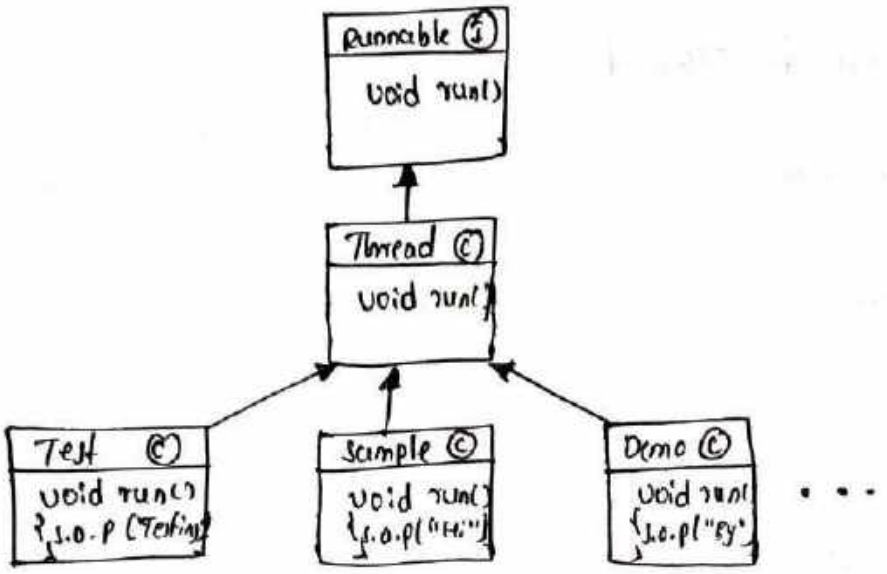
- "Whenever we want to pause the task some time & resume it and start the task again then we should go for thread.

RUNNABLE :-

- Runnable is an functional Interface, which belongs to java.lang package.
- Runnable Interface has run method.
- we can override run method, In two ways.
  - Extends
  - implements.
- by extending thread class.
- by implementing runnable Interface.



MULTI THREADING



" Processing multiple Threads simultaenously is called as multi Threading.

① Test

```
Test   t1=new Test();
Thread th1=new Thread(t1);
       th1.start();
```

② Sample

```
Sample   s1=new Sample();
Thread   th2=new Thread(s1);
         th2.start();
```

③ Demo

```
Demo   D1=new Demo();
Thread th3=new Thread(D1);
       th3.start();
```

program :-

```
package thread_topic;

class Test extends Thread
{
   public void run()
   {
     s.o.p("Hello");
   }
}
class sample extends Thread
{
   public void run()
   {
    s.o.p("This is Thread");
   }
}

class Demo ~~extends~~ implements Runnable
{
  public void run()
  {
   s.o.p("This is multi Threading");
  }
}

  class Main class
  {
    public static void main(String[] args)
    {
       Test t1=new Test();
       Thread th1=new Thread(t1);
              th1.start();

       Sample s1=new sample();
       Thread th2=new Thread(s1
              th2.start();

       Demo d1=new Demo();
       Thread th3=new Thread(d1)
              th3.start();
    }
  }
```

o/p

```
Hello
This is Thread
This is multi Threading.
```

Thread has Three Major properties :-

1] **Thread Id :-** It is generated by the system.

2] **Thread Name :-** It is by default thread name will be main. and we can change the thread name.

3] **Thread priority :-** It is a given priority given to the thread. where 1 is the least priority. & 10 is the maximum priority. & default priority will be 5.

current thread () :-

" current thread is a static method of Thread class, which will return the instance of currently executing thread."

```
class Thread
{
  static Thread current
  {
    thread()
  }
}
```

```
package thread_topic;
class Demo
{ public static void main (String[] args)
  {
    Thread th1 = new Thread.currentThread();
    S.o.p (th1.getId());
    S.o.p (th1.getName());
    S.o.p (th1.getPriority());
  S.o.p("- - - - * * * * * - - - - ");
        th1.setName ("Dinga Thread");
        th1.setPriority (8);
    S.o.p(th1.getId());
    S.o.p(th1.getName());
    S.o.p (th1.getPriority());
  }
}
```

O/P

```
1
main
5

1
Ding Thread
8
```

MULTI-THREADING PROGRAM :-

```
package thread topic;

class Demo extends Thread
{
    public void run()
    {
      for (int i=0; i<=10; i++)        // it's have interrupted exception
      {                                   try & catch it's automatically taken
        s.o.p (i);                        by system.
      }Thread.sleep (2000);
    }
}

class Sample extends Thread
{ public void run()
  { for (int i=101; i<=110; i++)       // it's have interrupted exception
    {                                    try & catch it's automatically taken
      s.o.p (i);                         by system.
      Thread.sleep (1000);
    }
}

class Main class
{ public static void main (String[] args)
  {
    Demo d1=new Demo ();
    Thread th1=new Thread (d1);
          th1.start();

    Sample s1=new Sample ();
    Thread th2= new Thread (s1);
          th2.start();

  }
}
```

O/P

```
1
2
101
3
4
5
102
103
6
7
104
.
.
.
```

:- PROCESSING SYNCHRONIZED :-

" processing Thread one by one is called as "synchronized."

It's have methods :-

① wait() :- wait() is an a non-static method of object class. which will make threads wait() to access the resources.

② notify() :- notify() is an a non-static method of object class, which, it just notify / notifies a thread which is above to access the threads.

③ notifyall() :- It is an non-static method of object class, which will notify the all threads which is above to access the resources.

note :- all these methods are used in case of synchronized.

THREAD LIFE CYCLE :-

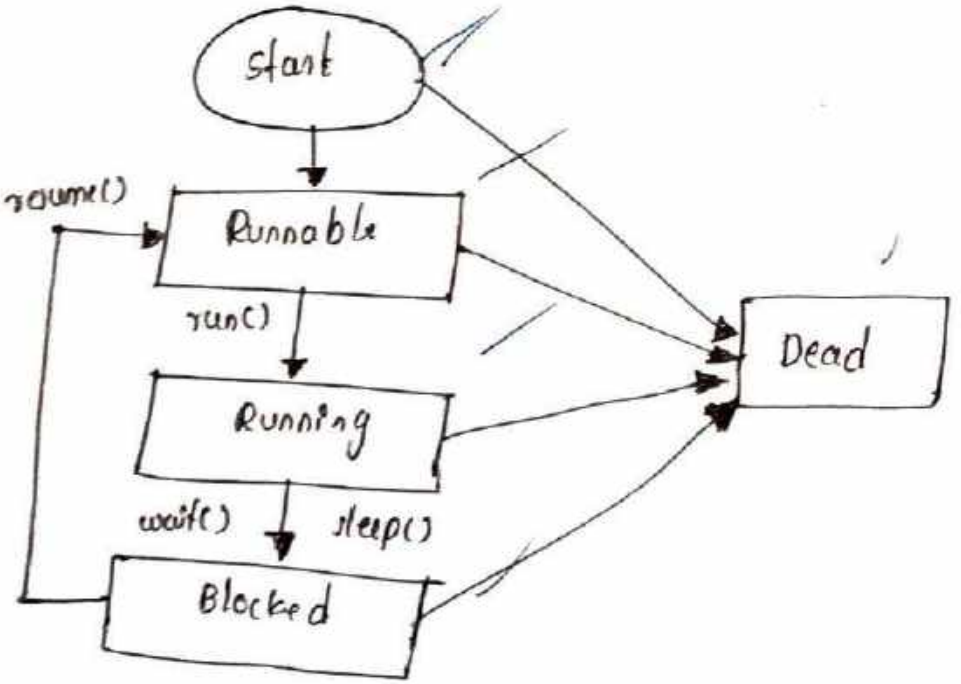


fig: Thread life cycle.

(1) It is a life cycle of any thread, where the thread starts and move to runnable state, and then, to running state, if thread goes for wait and sleep() it reaches blocked state and it can resume to runnable state, after some duration.

(2) at any of this stages the thread can reach dead state give to in-sufficient resources..

Differentiate b/w wait() and sleep() :-

wait()	sleep()
(1) wait is an non static method of the object class	(1) sleep is an static method of the ~~object~~ thread class.
(2) wait is use only in case of synchronized	(2) sleep it can be used in any code.
(3) wait() will make threads waits till receives notification	(3) sleep(), will pause the execution for few milisecond.

SYNCHRONIZED AND NOT SYNCHRONIZED MULTITHREADING :

Synchronized : if you want to not synchronized multithreading then remove the synchronized and add public.

```
class shared resource
{
  synchronized void res1()
  {
    for(int i=1; i<=10; i++)
    {
      s.o.p(i);
      Thread.sleep(1000);
    }
  }
  synchronized void res2()
  {
    for(int i=101; i<=110; i++)
    {
      s.o.p(i);
      Thread.sleep(2000);
    }
  }
}
class Demo extends Thread
{
  shared resource sr;

  Demo (shared resource sr)
  {
    this.sr=sr;
  }
  public void run()
  {
    sr.res1();
  }
}
```

// it have Interrupted Exception still take automatically.

// it have Interrupted Exception still take automatically.

```
class sample extends Thread
{
    shared resource sr;

    sample (shared resource sr)
    {
        this. sr = sr;
    }
    public void run ():
    {
        sr. res2();
    }
}

class main class
{
    public static void main (string[] args)
    {
        shared resource sr=new shared resource ();
        Demo d1=new Demo (sr);
        Thread th1=new Thread (d1);
                th1. start ();

        sample s1=new sample (sr);

        Thread th2=new Thread (s1);
                th2. start ();

    }
}
```

22/4/2023

:: COMPARABLE AND COMPARATOR ::

(R) **Comparable** :- comparable is one of the Interface.

(X) comparable is used to modify the objects sorting the objects in assending order. to sort the objects.

Example :-

```
class Employeess implements comparable {
  int id;
  String name;
  String city;

  @override
  public String toString() {
    return "Employeess [id:"+id+", name:+"name", city=+"city"]
  }

  @overridie
  public int compareTo (object o) {
    Employeess e=(Employeess) o;
    return this.city.compareToIgnoreCase (e.city);
  }      // (e.city.compareToIgnoreCase (this.city);

public class Mainclass
{
  public static void main (String [] args)
  {
    Employeess e1= new Employeess (1, "Nagu", "ILKAL");
    Employeess e2= new Employeess (5, "Manju", "Bagalakot");
    Employeess e3= new Employeess (4, "Manu", "UK");
    Employeess e4 = new Employeess (3, "Deepa", "Bg");
    Employeess e5= new Employeess (2, "Shiva", "UK");
```

```
Employeeiss[] arr = { e1, e2, e3, e4, e5 };
 arrays.sort(arr);
for (Employeeiss o : arr)
{
    S.o.p(n (o);
}
}
}
```

O/P

sorted

{id=1, name=Deepa, city=vx}

COMPARATOR :-

(1) COMPARATOR is one of the Interface.

(2) Comparator whenever, the class have Declare with final we go for comparator.

(3) Whenever the class is final we use comparator.

THANK
YOU