

★ INTRODUCTION →

- 1) History of JAVA
- 2) Variables -
- 3) Methods -
- 4) Condition statements
- 5) Loop statement
- 6) Array
- 7) String function
- 8) Static -
- 9) Non-Static & JVM Memory. ★ 1st MOCK.
- 10) Class & Object
- 11) Programming :-
 - Ⓐ Factorial
 - Ⓑ fibonacci series
 - Ⓒ Reverse String.
 - Ⓓ Prime number
 - Ⓔ odd numbers even
 - Ⓕ patterns
- 12) BLOCKS
- 13) Constructor -
- 14) Pass by value
- 15) Pass by reference
- 16) Composition
- 17) Inheritance
- 18) Method over loading ★ 2nd MOCK.
- 19) Method overriding
- 20) Type casting ✓
- 21) Polymorphism ✓
- 22) Abstract class ✓
- 23) Interface ✓
- 24) Package ★ PRE FINAL Mock
- 25) Access Specifier
- 26) Incapsulation
- 27) collection framework,
- 28) Object class
- 29) String class
- 30) Exception handling
- 31) Threads
- 32) file handling ★ FINAL Mock.

Q → What is JAVA?

Ans → Java is a object oriented programming language. which is used to develop the software.

Q → What is variable?

Ans → Variable is a named, memory, allocation. which is used to store some value or data. which can change n no of time during execution.

Syntax :-

datatype variable name = value;

int x = 10;

Ex :- plastic bottle = water;

Q → What is Method?

Ans → Method is a block of statement. which will get executed whenever is called invoke.

Q → What is class?

Ans → Class is a blueprint or template to create an object.

Ex -

Class

Sample

Body → $\left[\begin{array}{l} \{ \\ \rightarrow \text{PSVML} - \\ \{ \\ \} \end{array} \right] - \text{soul}$

Q → What is object?

Ans → Object is real time entity. which is having its own behaviour.

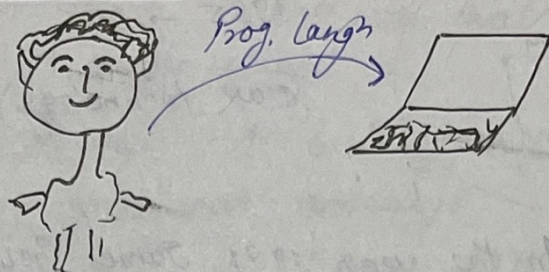
Q → List out the OOPS concept?

Ans → ① Inheritance ② Polymorphism ③ Incapsulation
④ Abstraction ⑤ class & Object.

★ Language →

language is a medium to communicate b/w two entities.

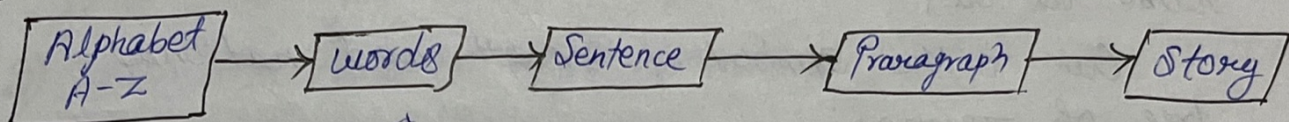
★ PROGRAMMING LANGUAGE →



It is a medium to communicate b/w human and machine.

Q → How the language is built?

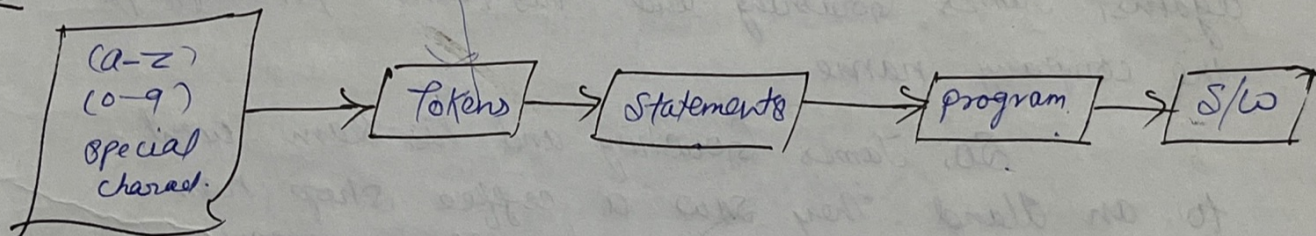
Ans →



Date - 25/08/2022

Q → How the programming language built?

Ans →



★ Softwares required →

① Editors

↳ Editplus ✓
↳ Notepad

② IDE (Integrated Development Environment)

60% ✓
Eclipse - 70%
NetBeans - 60%

③ Execution Area

cmd

③ JDK (Java Development Kit)

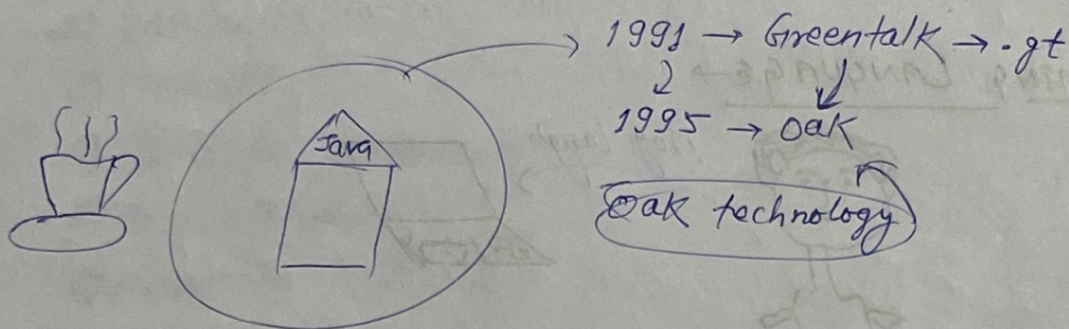
JDK 1.0

JDK 1.8

JDK 1.18

1.) HISTORY OF JAVA →

James Gosling



In the year 1991 James Gosling and his team developed a software called Greentalk with an extension of .gt.

In the year 1995 they renamed the s/w name as oak

Oak is the symbol of strength and national tree of Germany.

In the same year there is an existing company called oak technology. They file a case against James Gosling and his team for copying the company name.

So, James Gosling and his team went to an island. They saw a coffee shop named JAVA. They kept a software name JAVA and coffee mug as a logo of JAVA software.

Q → How to launch a Notepad?

Ans →

1.) On the system.

2.) Go to the left corner bottom and click on Search and write Notepad.

3.) Double click on the Notepad icon, Notepad will be launched.

Q → How to launch the Editplus?

Ans → 1) On the system.

2) Go to the browser and search editplus.

click on the first link scroll slowly. there you will find editplus link. click on that link it will navigate to the google drive.

3) click on downloads, it will give you the popup. click on the pop-up downloads anyways.

4) your Editplus will be downloaded.

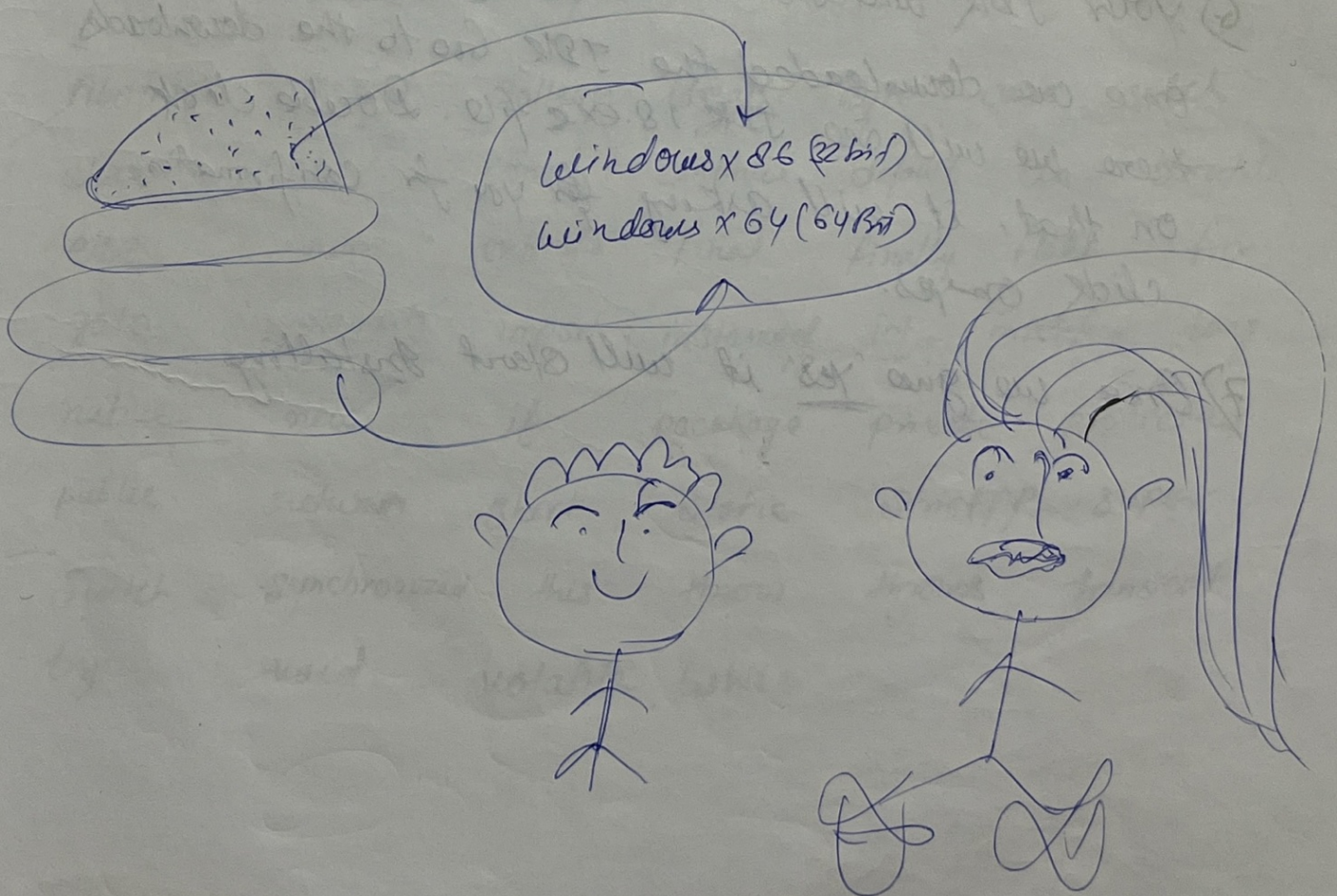
Date-26/08/22

Q → How to download and install JDK 1.8?

Ans → 1) On the system.

2) Go to the browser, type download JDK 1.8 and click on search.

3) Go to the first link scroll slowly. we will see windows x86^(32 bit) and windows x64 (64 Bit).



Q → How to check my system is whether 32bit or 64Bit?

Ans - 1) Go to this PC right click on it, click on properties.

2) There we will get to see our system type is 32 bit or 64 bit.

3) If my system is 64 bit then I will download windows x 64, My system is 32 bit then I will download windows x 86.

4) Click on download, it will give the pop-up, check the check box and click on enter. It will navigate to oracle.

5) Create a account and once again come to the login page give your user name and password which you have created and click on login.

6) Your JDK will started downloaded.

Once we downloaded the JDK, Go to the downloads, there we will see JDK 1.8.exe file. Double click on that, it will asking for you for confirmation. click on yes.

7) Once we give yes it will start installing.

★ TOKENS →

Definition → Tokens are the smallest individual part of the program.

→ In tokens we have —

1. Identifiers
2. Keywords
3. Operators
4. Separators
5. Comments
6. Literals

(1) Identifiers → Identifiers is a name given to JAVA Programs.
 • Keywords can not be used as identifiers.

(2) Keywords →

Keywords are the predefined words which has their own meaning.

We have around 50 keywords in core JAVA.

Abstract	Assert	Boolean	Break	Byte	case	catch
char	class	const	continue	default	do	double
else	enum	extends	final	finally	float	for
goto	implements	import	instanceof	int	interface	long
native	new	if	package	private	protected	
public	return	short	static	strictfp	super	
Switch	synchronized	this	throw	throws	transient	
try	void	volatile	while			

③ Operators →

Operators are the symbol which is used to perform ^{operations} on operands

Ex- $10 \oplus 10$
 $\swarrow \searrow$
 operands

We have some types -

<u>operator type</u>	<u>category</u>	<u>Precedence</u>
1) Arithmetic	Postfix Prefix	Expr++, Expr--, ++Expr, --Expr, +Expr, -Expr, !=
2) Relational	comparison-Equality	<, >, <=, >=, instance of ==, !=
3) Bitwise	Bitwise AND, Bitwise exclusive OR Bitwise inclusive OR	& ^
4) Logical	Logical AND Logical OR	&&
5) Ternary	Ternary	? : :
6) Assignment	Assignment	+=, -=, *=, /=, %=, &=, ^=, =, <<=, >>=, >>>=

4) Seperators + separators are used to separate the JAVA code.

() → Paranthesis

{ } → Braces

[] → Brackets

; → Semicolon

,

: → Colon

⑤ COMMENTS → Comments are used to provide the extra information inside the JAVA program.
We have two types of comments-

① Single line comment

Syntax → // _____

② Multi line comment (Blockline Comment)

Syntax → /* _____

_____* /

⑥ LITERALS → Literals are the values used in JAVA to perform some operations.
In literals we have —

1) Number literals — integer literals 0, 100, 1000
— decimal literals 100.0, 99.99, 50.09

2) Character literals → 'A', '9', 'x', ''

3) String literals → "", "Anil \$ 99.9", "A", "99"

4) Boolean literals → true, false

1/1

Date - 31/08/21

Q. Write a prog. to print HELLO JAVA.

Ans.

```
class Dinga {  
    public static void main (String [] args) {  
        System.out.println ("HELLO JAVA");  
    }  
}
```

Identifier
Class Identifier
Main method declaration
Printing Statement
End of Statement
Class body definition
main body definition

Date -

★ Commands used in CMD →

cls - clearscreen

cd - Change directory

cd.. - change directory from current to previous

javac - java compilation

java - java interpretation

★ JAVA ARCHITECTURE →

As a software engineer we will write a program in notepad or edplus.

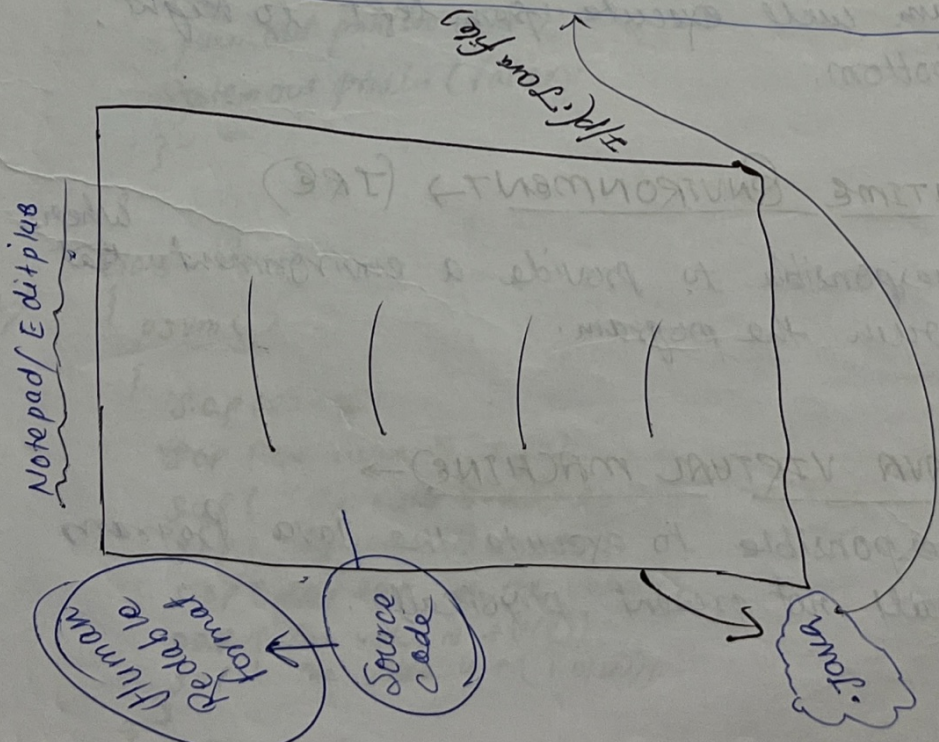
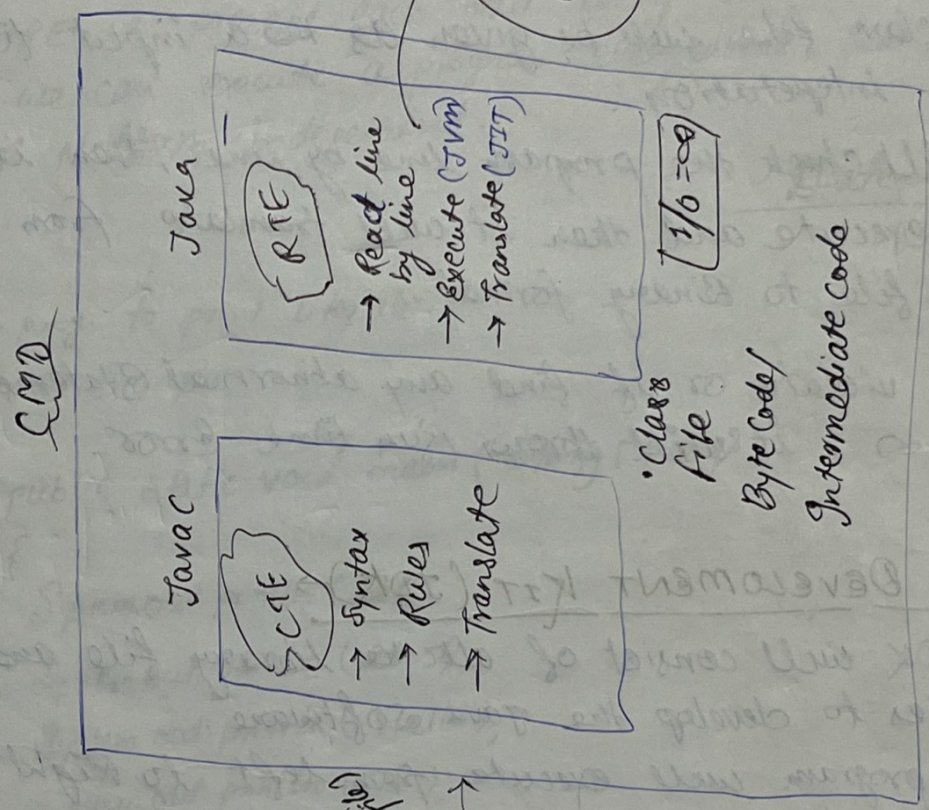
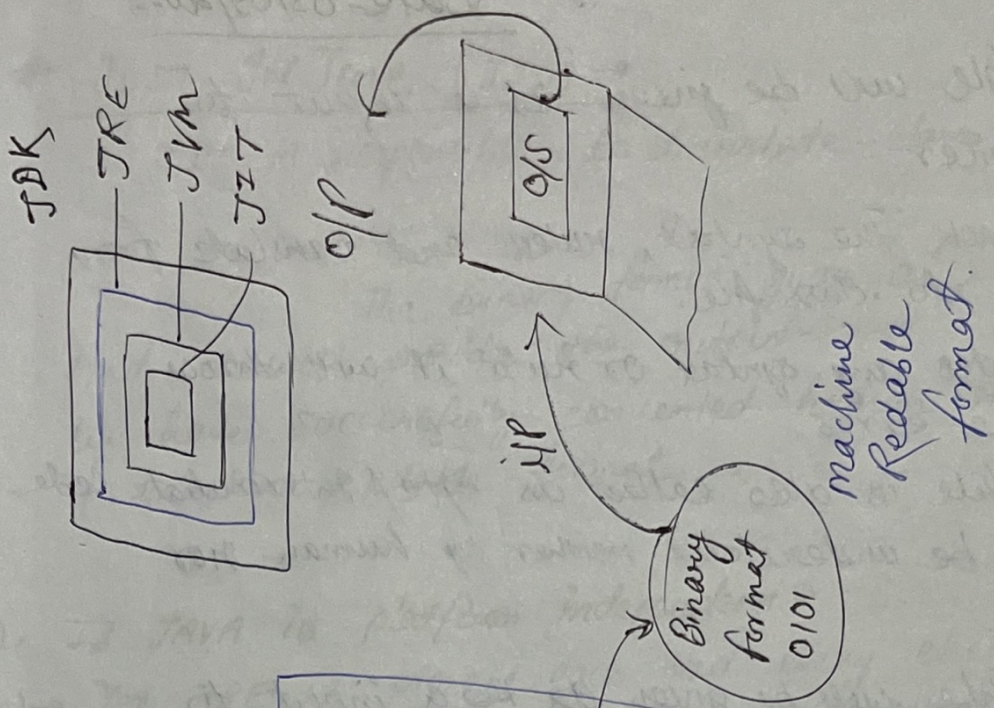
which will be in human readable format.

This program is also called as source code, once we write a program we will save the program with an extension of .java.

In command prompt we have two operations.

① JAVA C (Java Compilation)

② JAVA I (Java Interpretation)



- The .java file will be given as a input to java compiler.
- It will check the syntax, rules and translate from .java file to .class file.
- If we violates any syntax or rule it will throw compile time error.
- This .class file is also called as byte/intermediate code. It can not be understood neither by human nor by machine.
- This .class file will be given as a input to java interpretation.
- It will check the program line by line, then it will execute and then it will translate from .class file to Binary format.
- If we violate or if find any abnormal Statement $1/0 = \infty$, it will throw Run time error.

* JAVA DEVELOPMENT KIT (JDK) →

JDK will consist of all the library file and utilities to develop the java software.

The program will execute from left to Right, top to bottom.

* JAVA RUNTIME ENVIRONMENT → (JRE)

JRE responsible to provide a environment ^{where} that we can run the program.

* JVM (JAVA VIRTUAL MACHINE) →

JVM is responsible to execute the Java Program But it will not present physically.

★ JUST IN TIME (JIT) →

JIT is responsible to translate class file to Binary format.

The Binary format given as a input to OS.
And OS will provide the output.
We have successfully converted human readable format to machine Redable format.

Q. Is JAVA is platform independent?

Ans → Yes, JRE is present each and every electronic device.
Hence we can execute a program everywhere that's why Java is platform independent.

Date-05/09/2022

Q. Write a prog. to print literals.

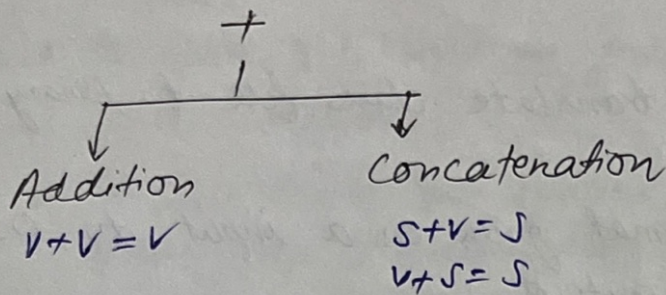
Ans → class Sample

```
{  
    public static void main (String[] args)  
    {  
        System.out.println (20);  
        System.out.println (20.54);  
        System.out.println ("Hello");  
        System.out.println ('A');  
        System.out.println (false);  
    }  
}
```

Q → Q

Class Sample

```
{  
    public static void main (String[] args)  
    {  
        S.O.P (20+20);  
        S.O.P ("The value is " + 20);  
        S.O.P ("The value is " + 20 + 20);  
        S.O.P ("The value is (20+20);  
        S.O.P (20+20 + " is the value");  
        S.O.P ("The value is + (1/2)");  
        S.O.P ("The value is + (1.0/2)");  
    }  
}
```

concatenation is used to merged between two strings

if $\frac{1}{2} = 0.5$

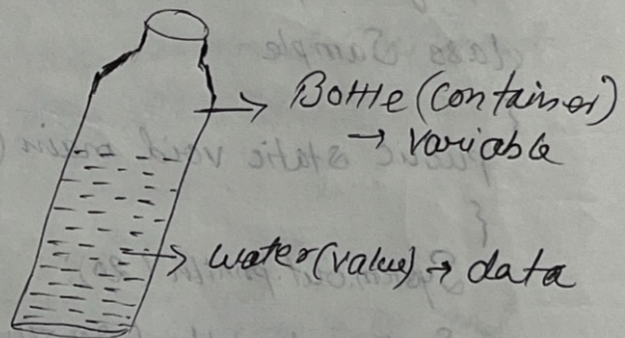
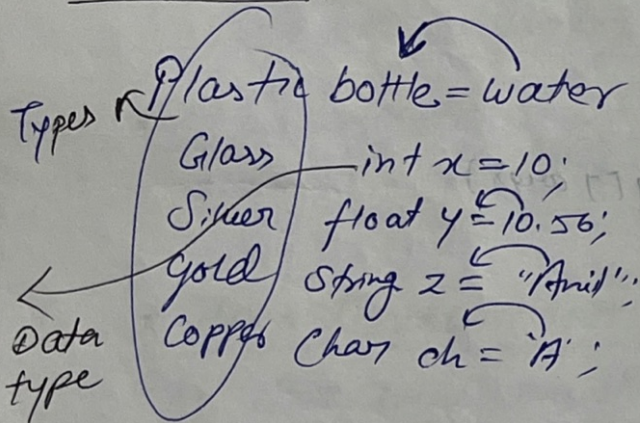
There is no decimal in num & den.

it will give 0 only according to Java.

$\frac{1.0}{2} = 0.5$ (there is decimal)

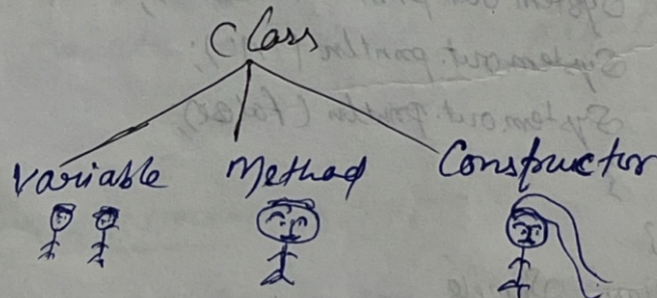
Date → 06/09/2022

★ VARIABLE →



Java Types

Class
Enum
Annotation
Interface



Variable is a named, memory, allocation which is used to store some value or data. ~~which~~ can store and it can change ~~no.~~ of time during execution.

VARIABLEPrimitive datatypes

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

Non-Primitive datatypes

- 1) String
- 2) Array
- 3) ref. type
- 4) class type

① VARIABLE DECLARATION →

Syntax:- datatype variable-name;

Ex → int x;

② VARIABLE INITIALIZATION →

Syntax:- variable-name = value;

Ex → x = 10;

③ VARIABLE UTILIZATION →

System.out.println(x);

O/p → 10

④ Variable declaration and Initialization in a single line

Syntax:- datatype variable-name = value;

Ex → int x = 10;

⑤ Variable Re-Initialization →

Ex → int x = 10;

Sop(x); O/p →

x = 50;

Sop(x); O/p → 50

⑥ Copying the value from one variable to another →

```
int x = 20;  
int y = x;
```

Date - 09/09/22

* Classification of Variables →

Variables are classified into two types.

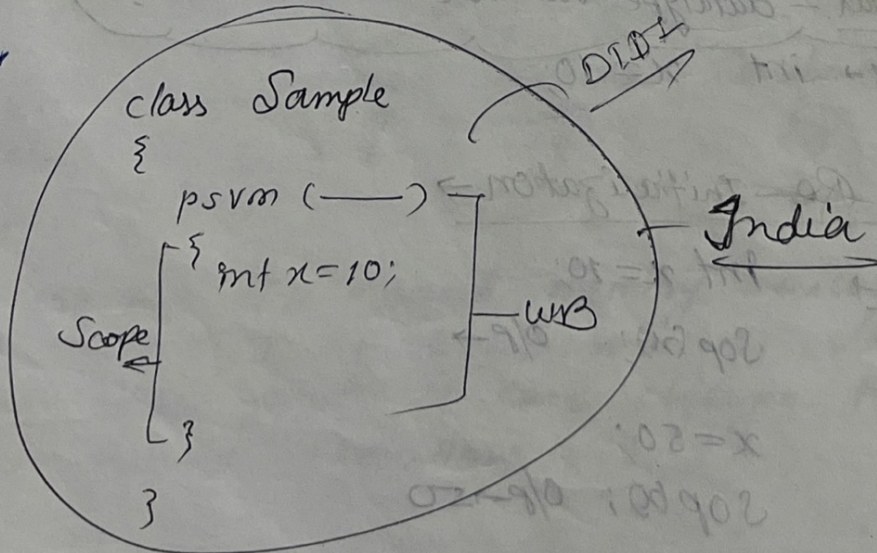
- ① Local Variables.
- ② Global Variable.

① LOCAL VARIABLES →

Any variable which is declared inside a method or within the method is called as Local Variable.

- The scope of the local variable is from beginning of the method till end of the method.
- Local variable will not have default values.
- It cannot be classified into static and non-static.
- Without declaration and initialization we can not utilize the variable.

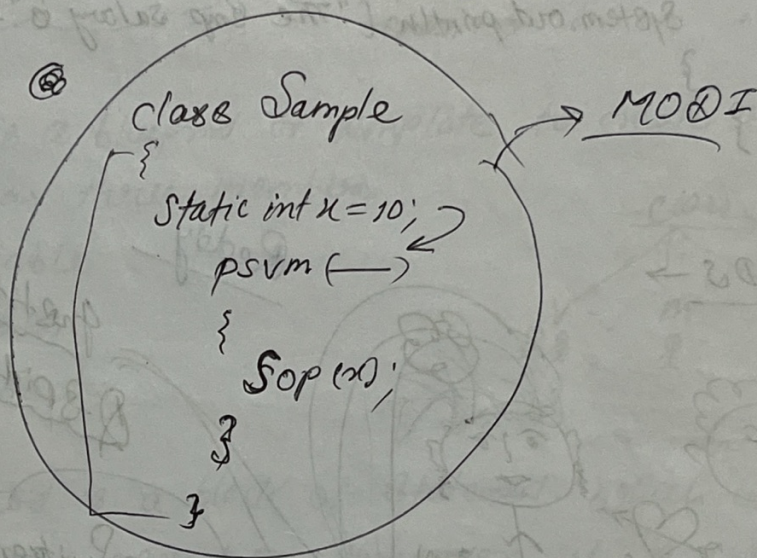
~~Method~~



② GLOBAL VARIABLE →

Any variable which is declared outside the method inside the class is called as global variable.

- The scope of the global variable is from beginning of the class till end of the class.
- Global variable will have default values.
- It can be classified into static and non-static.
- Once we declared the global variable immediately on the next line we can not re-initialized.



* Default values:

byte → 0

Short → 0

int → 0

long → 0

Float → 0.0f

double → 0.0d

char → 'u/0000' - (unicode)

boolean → false

String → null

Q- Write a prog to print emp id and emp name and emp salary of a given emp. Date
10/09/22

Ans

class Employee

{

psum(____)

{

int Emp-id = 1;

String Emp-name = "Tom";

double Emp-salary = 50000.00;

System.out.println("The Emp id is " + Emp-id);

System.out.println("The Emp name is " + Emp-name);

System.out.println("The Emp salary is " + Emp-salary);

}

}

★ METHODS →

Syed

Mechu

LOVE

CBU

Reddy

Institute

Spider

IBM

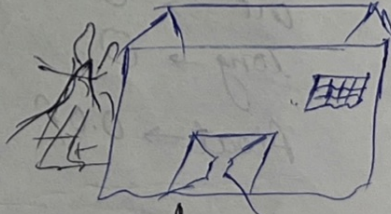
GOK

Switzerland

India

18th Anniversary

Switzerland



India

2nd Anniversary
Hooghly water

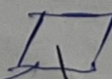
Switzerland



THJ
man

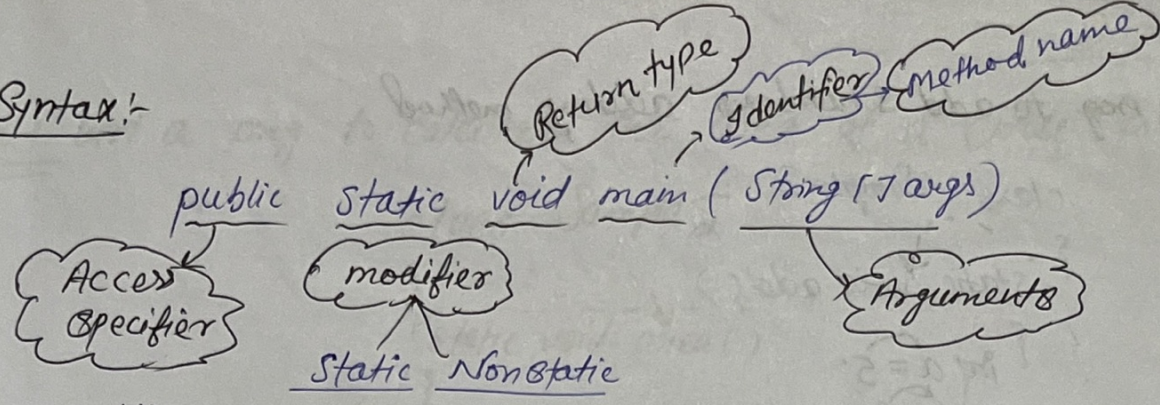
India

99th anniversary



containers (to contain figures)

Syntax:-



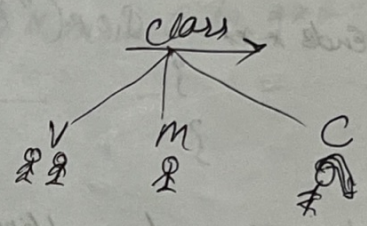
- public
- private
- package
- protected

void → it will return nothing
 int → it will return int.
 String → " " " String.
 boolean → " " " true/false
 double → " " " double
 float → " " " float
 char → " " " char

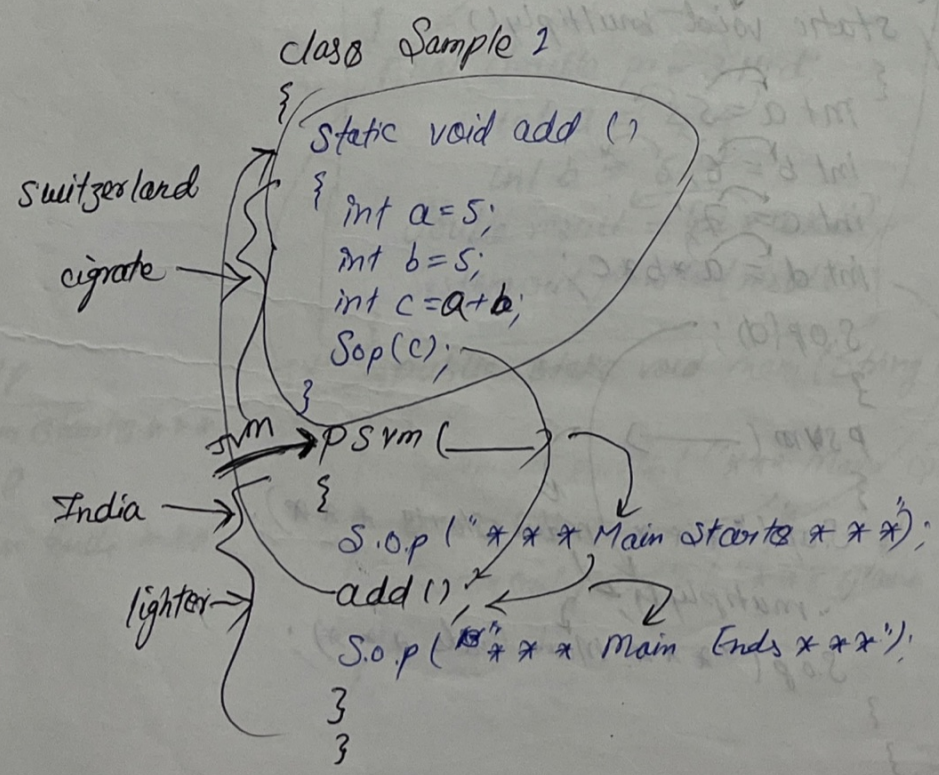
class type → object

★ Class → class is a blueprint or template to create an object. Class has three members.

- ① Variable
- ② Member
- ③ Constructor



★ Method → method is a block of Statement. which is used to perform some operation. whenever it is called or invoke.



*** Main Starts ***
 10
 *** main Ends ***

Q. Wrt. a prog. to add 3 numbers adding method.

~~Ans~~

```

class Sample3
{
    static void add()
    {
        int a = 5;
        int b = 6;
        int c = 7;
        int d = a + b + c;
        S.o.p(d);
    }
}

```

↓ JVM

```

PSUM (
{
    S.o.p("*** main starts ***")
    add()
    S.o.p("*** Main Ends ***")
}
}

```

O/P

*** Main Starts ***
18
*** Main Ends ***

Q. write a prog. to multiply 3 no. using methods.

~~Ans~~ void multiply

```

class Sample4
{
    static void multiply()
    {
        int a = 5;
        int b = 6;
        int c = 7;
        int d = a * b * c;
        S.o.p(d);
    }
}

```

↓ JVM

```

PSUM (
{
    S.o.p("*** main starts ***")
    multiply()
    S.o.p("*** Main Ends ***")
}
}

```

O/P

*** Main Starts ***
210
*** Main Ends ***

Q1 Wrt a prog. to calculate the area of a circle using Method

class Sample2

```
{
    static void area()
    {
        final double pi = 3.142;
        int r = 5;
        double result = (pi * r * r);
        sop(result);
    }
}
```

Switzerland

$$= \pi r^2$$

JVM

psvm()

System.out.println("*** Main Starts ***");

area();

System.out.println("*** Main Ends ***");

}

}

O/P
*** Main Starts ***

78.5

*** Main Ends ***

India

Q2 Area of Ellipse:

class Sample3

```
{
    static void area()
    {
        final double pi = 3.14;
        int a = 4;
        int b = 5;
        double result = (pi * a * b);
        System.out.println(result);
    }
}
```

JVM

public static void main(String[] args)

{ System.out.println("*** Main Starts ***");

area();

System.out.println("*** Main Ends ***");

}

}

O/P
*** Main Starts ***

62.8

*** Main Ends ***

★ Method with Parameters →

Whenever we want to give input to the method than we go for parameters.

method with

class Sample 7

```
{
    static void add (inta, intb)
```

```
{
    int c = a+b;
```

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

→ JVM → public static void main (String [] args)

```
{
    System.out.println ("*** Main starts ***");
```

```
    add (10, 5);
```

```
    System.out.println ("*** Main Ends ***");
}
```

```
}
```

O/P

*** Main starts ***

15

*** Main Ends ***

②

class Sample 8

```
{
    static void add (inta, intb, int c)
```

```
{
    int d = a+b+c;
```

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

→ JVM → public static void main (String [] args)

```
{
    System.out.println ("*** Main starts ***");
```

```
    add (10, 5, 6);
```

```
    System.out.println ("*** Main Ends ***");
}
```

```
}
```

```
}
```

O/P

*** Main starts ***

21

*** Main Ends ***

Q. wrt a prog. to calculate multiplication of 3 no. using method with parameters.

class Multiplied

```
{
    static void multiply (inta, intb, intc)
```

```
{
    int c = a * b * c;
```

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

sim → public static void main (String[] args)

```
{
    System.out.println ("*** main starts ***");
```

```
    multiply (5, 2, 3);
```

```
    System.out.println ("*** main ends ***");
}
```

```
}
```

O/P

*** main starts ***

36

*** main ends ***

Q. wrt a prog calculate area of circle using method with parameters.

class Circle

```
{
    static void multiply (double pi, int r)
```

```
{
```

```
    double c = double pi * r * r;
```

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

sim → public static void main (String[] args)

```
{
```

```
    System.out.println ("*** main starts ***");
```

```
    multiply (3.142, 8);
```

```
    System.out.println ("*** main ends ***");
}
```

```
}
```

```
}
```

O/P

201.088

★ Method with return type :-

When we want an operation output for further operation then we go for method with return type.

```
class Sample9
```

```
{
    static int add()
    {
```

```
        int a = 5;
        int b = 10;
        int c = a + b;
        return c;
    }
```

```
}
psvm (—)
```

```
{
    Sop ("*** Main Starts ***");
```

```
    int x = add();
```

```
    Sop (x);
```

```
    Sop ("*** Main Ends ***");
}
```

```
}
```

Manager

HR

HR

Home

Man

IBM

60K

Dingy

+5K

Dingy

+10K

Sum

```
{
```

```
    Sop ("*** Main Starts ***");
```

```
    int x = add();
```

```
    Sop (x);
```

```
    Sop ("*** Main Ends ***");
}
```

```
}
```

int result = x + 5000

Sop (result);

★

```
class Dingy
```

```
{
```

```
    static double managerhike ()
    {
```

```
        double basicpay = 25000.00;
```

```
        double homeallowance = 25000.00;
```

```
        double foodandtravel = 10000.00;
```

```
        double pay = basicpay + homeallowance + foodandtravel;
```

```
        return pay;
    }
```

```
}
psvm (—)
```

```
{
    Sop ("*** Main Starts ***");
```

```
    double salary = managerhike ();
```

```
    double increment = salary + 5000.00;
```

```
    Sop (increment);
```

```
    Sop ("*** Main Ends ***");
}
```

```
}
```

65000

830.108

Q- Wrt to prog. to calculate adding 3 nos. ~~with~~ using method with return type.

```
class Sample
{
    static int add()
    {
        int a=5;
        int b=5;
        int c=5;
        int d = a+b+c;
        return d;
    }
}
```

```

sum → psvm(——) → {
    sop("*** Main Starts ***");
    int x = add();
    int result = x-10;
    sop(result);
    sop("*** Main Ends ***");
}
}
```

O/P

*** Main Starts ***

105

*** Main Ends ***

Q:- Wrt to prog. to multiply 3 nos. using method with return type.

```
class Sample
{
    static int multiply()
    {
        int a=5;
        int b=5;
        int c=5;
        int d = a*b*c;
        return d;
    }
}
```

```

sum → psvm(——) → {
    sop("*** Main Starts ***");
    int x = multiply();
    int result = x+5;
    sop(result);
    sop("*** Main Ends ***");
}
}
```

O/P

*** Main Starts ***

125

*** Main Ends ***

Q → wrt to C++ to calculate area of circle by using method with return type.

★ Real time Example →

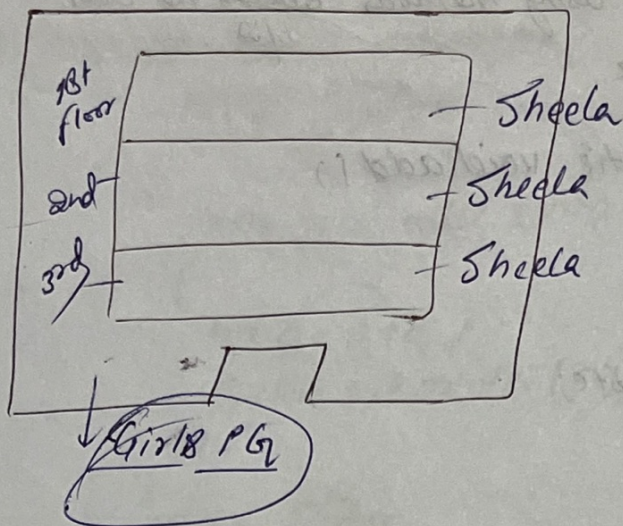
① Freeze ② Attendance punch info. ③ ATC

★ STATIC → Any member which is declared with a keyword static which is called static member of the class.

• RULES →

- Static member is always associated with class.
- Static is one copy.
- Static members will get store in static pool area.
- To access the static members from one class to another class, we use this syntax :-

class name. methodName();
class name. variableName();



Date-17/09/22

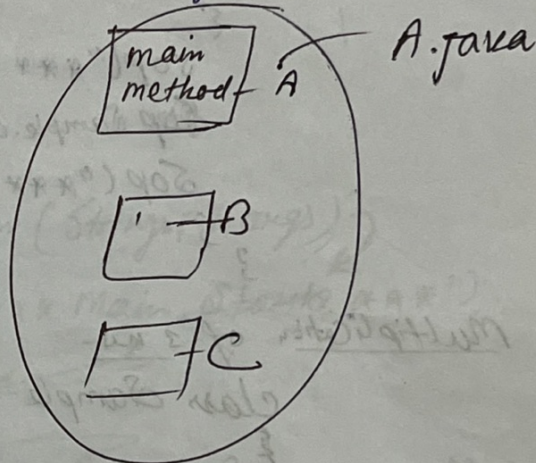
★

```
class Sample
{
    static int x = 10;
    static void hi()
    {
        S.O.P("Hello");
    }
}

// Main
public class Main
{
    public static void main(String[] args)
    {
        Sample s = new Sample();
        s.hi();
    }
}
```

O/P
10
Hello

NOTE → The class which is having a main method that should be save as a file name.



★ Wrt a prog to add two no. using methods between the classes.

```
class Sample
{
    add static void add()
    {
        int a = 5;
        int b = 10;
        int c = a + b;
        S.O.P(c);
    }
}

class Main
{
    public static void main(String[] args)
    {
        S.O.P("*** main Starts ***");
        Sample.add();
        S.O.P("*** main Ends ***");
    }
}
```

Main.java

~~Static~~ ^{Static} ~~with~~ ^{with}
* Write a prog to add 3 no. using methods ~~in~~ ⁱⁿ the class.
b/w

```
class Sample
{
int a=5; Static void add()
{
    int a=5;
    int b=10;
    int c=15;
    int d=a+b+c;
    Sop(d);
}
}
```

class Main1

```
{
    psum( )
{
    sop("*** main starts ***");
    Sop Sample.add();
    Sop("*** main Ends ***");
}
}
```

* Multiplication of 3 no.

class Sample

```
{
int Static void add Multiply()
{
    int a=5;
    int b=10;
    int c=15;
    int d=a*b*c;
    Sop(d);
}
}
```

class Main2

```
{
public Static void main (String[] args)
{
    sop("*** main starts ***");
    Sample.multiply();
    Sop("*** main Ends ***");
}
}
```


Q. wrap to add two no. using Static between the classes with method with parameter?

```

class Sample
{
    static void add (inta, intb)
    {
        int c = a+b;
        System.out.println(c);
    }
}

class Demo
{
    static void sub (intx, inty)
    {
        int z = x-y;
        System.out.println(z);
    }
}

class Main3
{
    public static void main (String[] args)
    {
        System.out.println("*** Main Starts ***");
        Sample.add (5, 5);
        Demo.sub (10, 5);
        System.out.println("*** Main Ends ***");
    }
}
  
```

O/P

```

*** Main Starts ***
16
5
*** Main Ends ***
  
```

Q. write a prog to multiply 3 no. by using Static between the classes with method with parameter?

Ans →

```

class Sample
{
    static void multiply (inta, intb, intc)
    {
        int d = a * b * c;
        S.op (d);
    }
}

class Main
{
    public static void main (String[] args)
    {
        S.op ("*** Main Starts ***");
        Sample.multiply (2, 2, 2);
        S.op ("*** Main Ends ***");
    }
}
  
```

O/P

```

*** Main Starts ***
8
*** Main Ends ***
  
```


Q → wrt a prog. to calculate area of circle by using static between the classes with method with parameters.

Ans →

```

class Sample
{
    static void area (double pi, int r)
    {
        double c = pi * r * r;
        sop (c);
    }
}

class main5
{
    public static void main (String[] args)
    {
        System.out.println ("*** Main Starts ***");
        Sample.area (3.14, 5);
        System.out.println ("*** Main Ends ***");
    }
}
    
```

O/P

```

*** main Starts ***
78.5
*** main Ends ***
    
```

Q → wrt a prog. to add 2 numbers using static b/w the classes with method with return type.

```

class Sample
{
    static int add ()
    {
        int a = 5;
        int b = 5;
        int c = a + b;
        return c;
    }
}

class main
{
    public static void main (String[] args)
    {
        sop ("*** Main Starts ***");
        int n = add ();
        int result = n + 5;
        sop (result);
        sop ("*** Main Ends ***");
    }
}
    
```

O/P

```

*** Main Starts ***
10
*** Main Ends ***
    
```


Q- Wrt a prog to add 3 no using static b/w the class with method with return type.

Class Sample

```
{
    static int add()
    {
        int a = 5;
        int b = 5;
        int c = 5;
        int d = a + b + c;
        return d;
    }
}
```

class main

```
{
    psum (—)
    {
        sop ("*** main starts ***");
        int x = Sample.add();
        sop(x);
        sop ("*** main ends ***");
    }
}
```

Q multiplication of 3 no

Class Sample

```
{
    static int multiply()
    {
        int a = 2;
        int b = 2;
        int c = 2;
        int d = a * b * c;
        return d;
    }
}
```

class main

```
{
    psum (—)
    {
        int x = Sample.multiply();
        sop(x);
    }
}
```

Q/P

8

Q7 Area of Trapezoid

$$\frac{1}{2}(a+b)h$$

class Sample

```
{
    static double area()
    {
        double v = 0.5;
        int a = 5;
        int b = 6;
        int h = 8;
        double d = v * (a+b) * h;
        return d;
    }
}
```

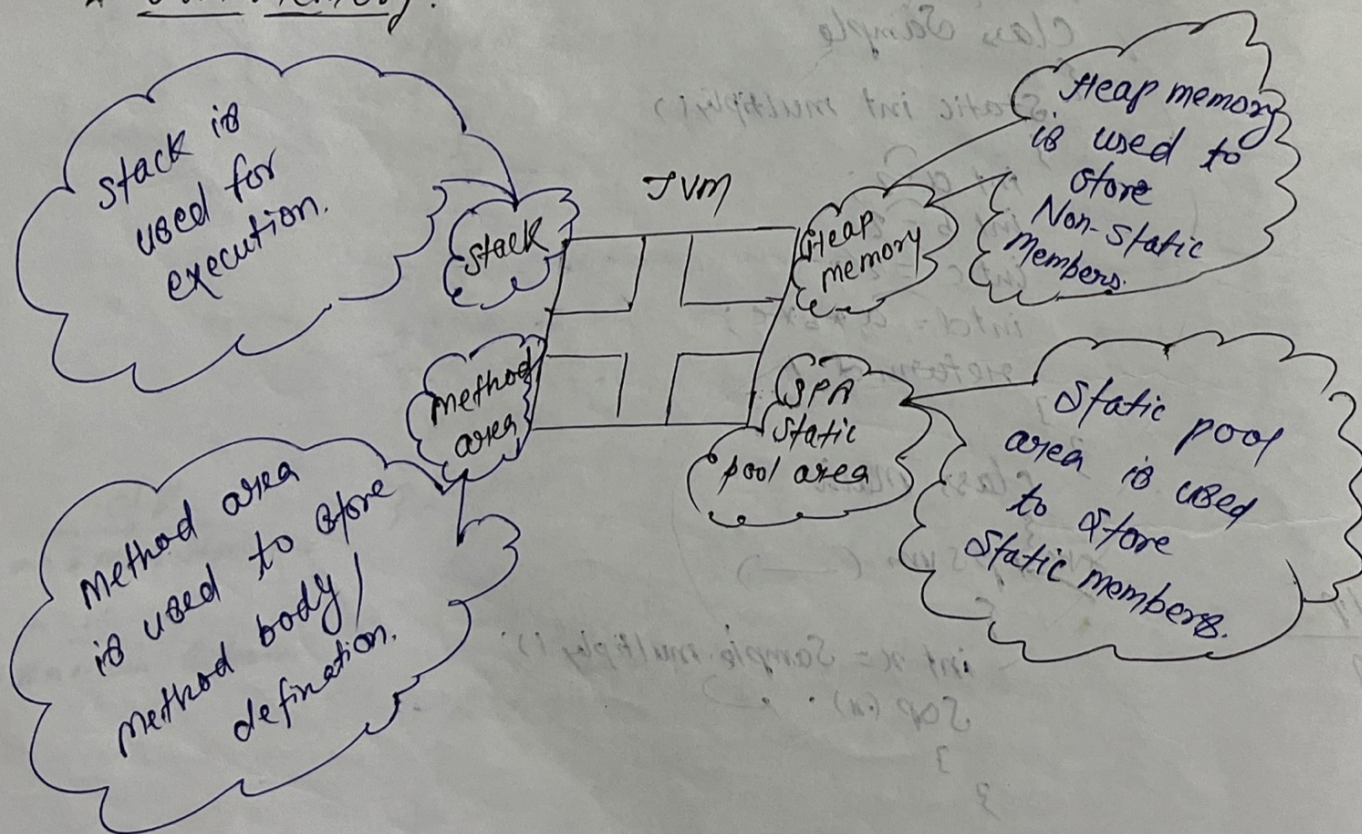
class main b

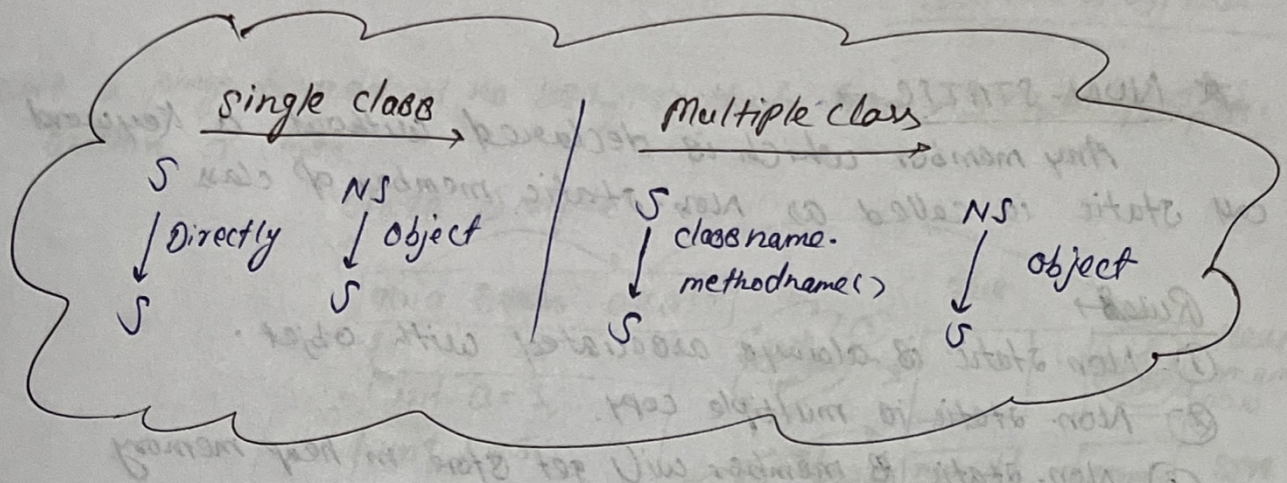
```
{
    psum ( )
    {
        int x = Sample.area();
        sop(x);
    }
}
```

O/P

Date → 21/09/2022

* JVM memory :-





Object

Operator

Constructor

New

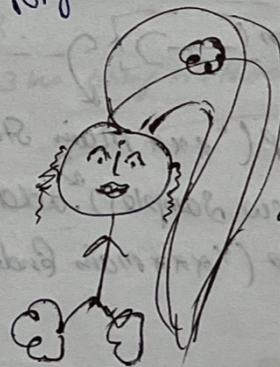
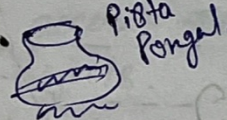
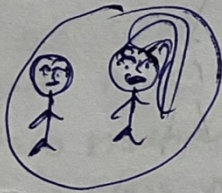
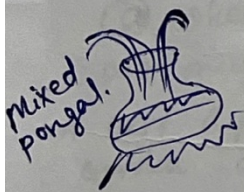
class_name()

New is an operator which will create random memory space in heap memory.

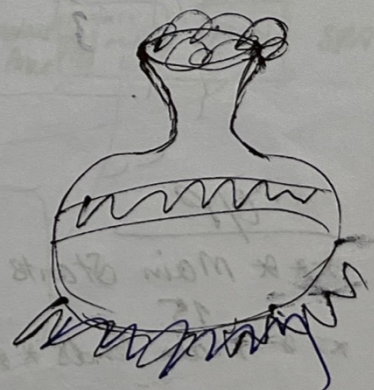
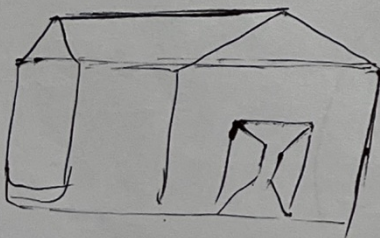
Constructor is used to initialize all the non static members in heap memory.

Date- 22/09/22

NOTE → Constructor Name Should be same as class Name.



Badam Pongal



★ NON-STATIC →

Any member which is declared without a keyword static is called as non-static member of class.

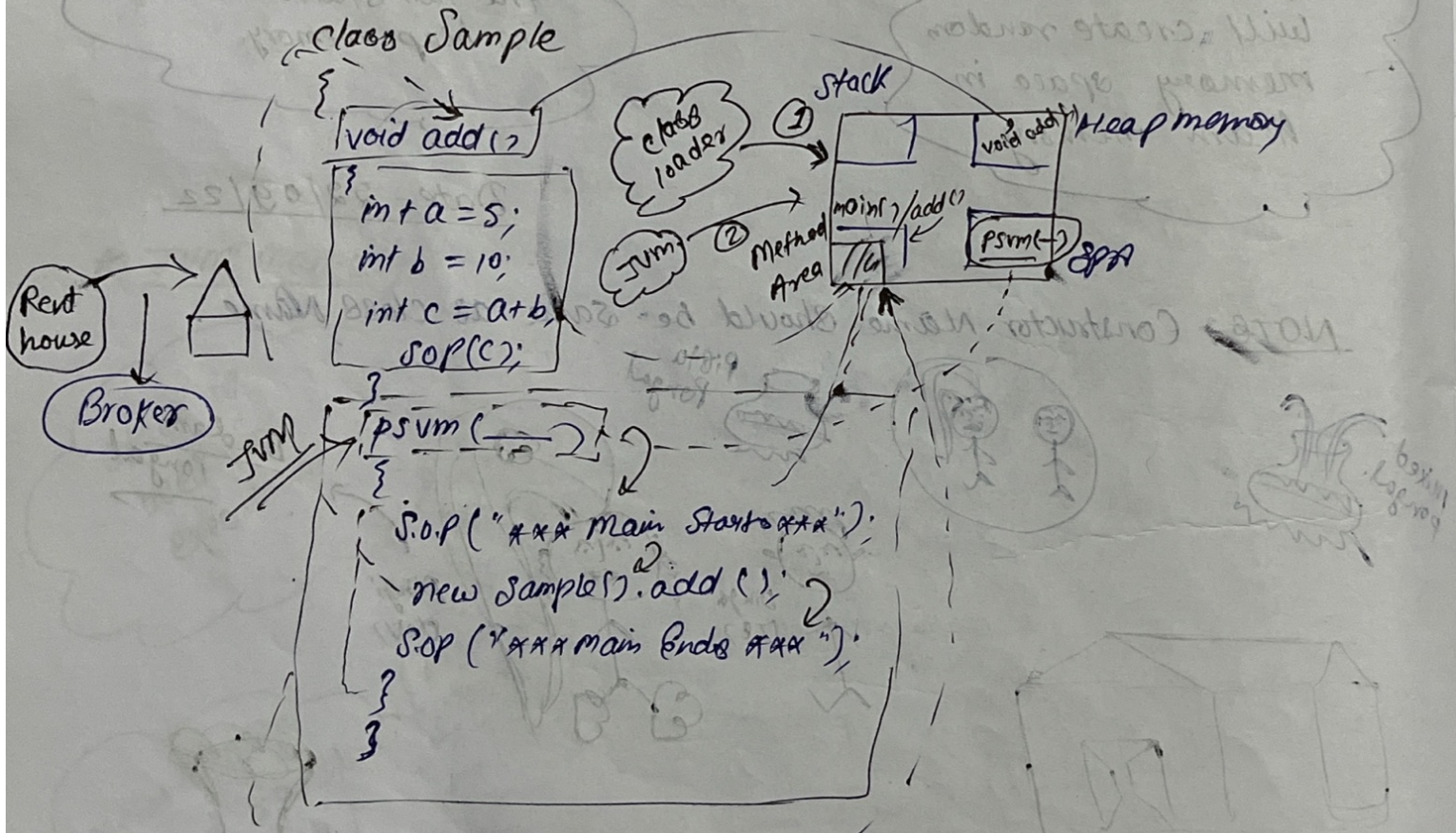
Rules →

- ① Non-static is always associated with object.
- ② Non-static is multiple copy.
- ③ Non-static member will get store in heap memory.
- ④ To access from the Non-Static to static

Syntax:-

Object.variableName
Object.methodName()

Date → 23/09/22



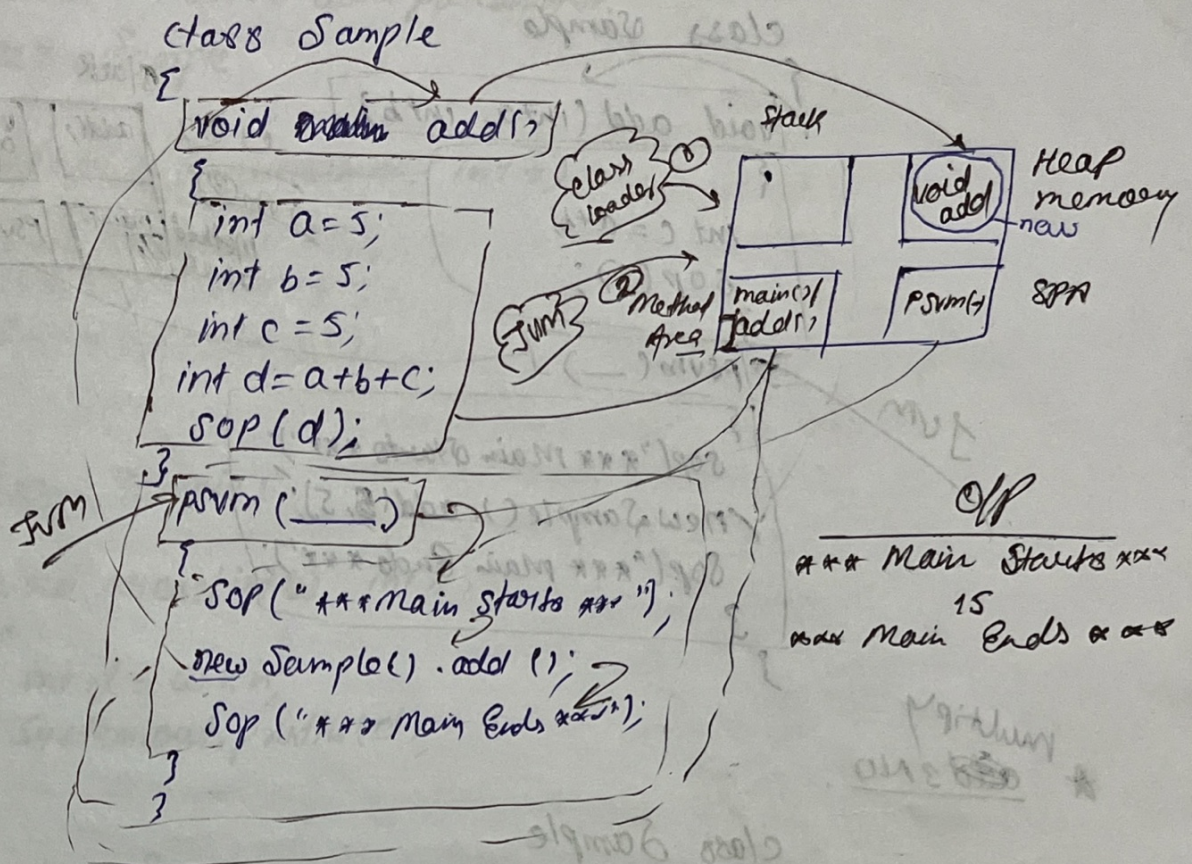
O/P

*** Main Starts ***
15
*** Main Ends ***

Date = 24/09/2022

* Wrt a prog. to add 3 no using methods from nonStatic to Static within a class.

Ans →

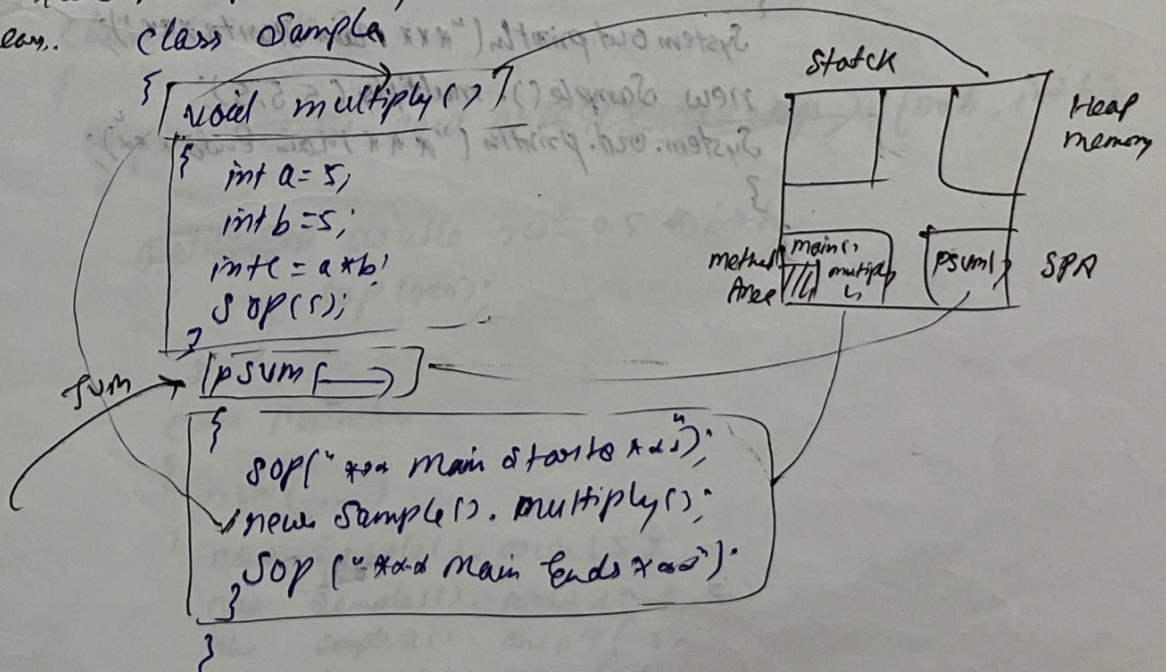


NOTE →

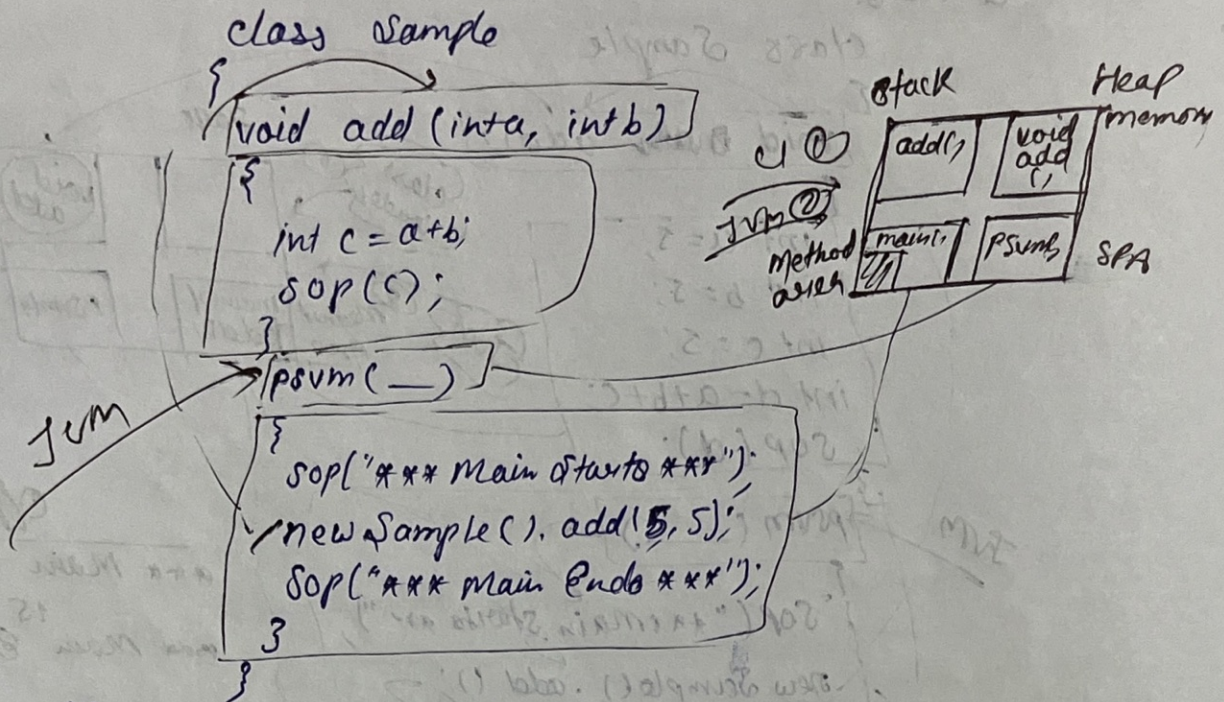
① When we want to access the same method for n no of times in a heap memory, it will create n no. of random memory space.

② Whenever we want to access same method for n no of times we must create n no. of objects.

* Wrt a prog. to multiplication of 3 no using methods from nonStatic to Static within a class.



* Wrt a prog to add 2 no with using method with parameters from non static to static within a class.



* multiply
add 3 NO.

class Sample

```

{
    void multiply(int a, int b, int c)
    {
        int d = a * b * c;
        System.out.println(d);
    }
    public static void main(String[] args)
    {
        System.out.println("*** Main Starts ***");
        new Sample().multiply(5, 5, 5);
        System.out.println("*** Main Ends ***");
    }
}

```


Date + 26/09/22

* Wrt a prog. to calculate area of circle, rectangle, triangle, square.
by using methods non-static to static ~~with~~ b/w the class ~~with~~
~~parameter~~ method with parameter.

Ans →

```
class Sample
{
    void area(double pi, int r)
    {
        final double pi = 3.142;
        double c = pi * r * r;
        System.out.println(c);
    }
}

class Sample1
{
    void area2(int w, int h)
    {
        int d = w * h;
        System.out.println(d);
    }
}

class Sample2
{
    void area3(int a)
    {
        int b = a * a;
        System.out.println(b);
    }
}

class Sample3
{
    public static void area4(String s) (int h, int b)
    {
        static void double res = 0.5 * b * h;
        sop(res);
    }
}

class Mainclass
{
    psum(—)
    {
        new Sample().area(5);
        new Sample1().area2(5, 5);
        new Sample2().area3(5);
        new Sample3().area4(5, 5);
    }
}
```


* Wrt a prog. to calculate addition of two no with the method as non-static by using return type. within the class

Class Sample

```
{
    int
    void add()
    int a = 5;
    int b = 5;
    int c = a + b;
    return (c);
}

public static void main (String[] args)
{
    int x = new Sample().add();
    System.out.println (x);
}
}
```

* Wrt a prog. to calculate the area of circle with method as non-static with return type within the class.

Class Sample

```
{
    double area;
    final double pi = 3.142;
    int r = 5;
    double res = pi * r * r;
    return res;
}

public static void main (String[] args)
{
    double x = new Sample().area();
    System.out.println (x);
}
}
```


* Wrt a prog to calculate area of circle, rectangle, square, triangle with the method as non static as by using return type between the class.

class Circle

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

class Rectangle

```
{
    int area()
    {
        int w = 5;
        int h = 5;
        int res = w * h;
        return res;
    }
}
```

class Square

```
{
    int area()
    {
        int a = 5;
        int res = a * a;
        return (res);
    }
}
```

class Square

```
{
    double area()
    {
        int b = 5;
        int h = 5;
        double res = 0.5 * b * h;
        return res;
    }
}
```

class main

```
{
    psvm ( )
    {
        double x = new Circle().area();
        sop(x);
    }
}
```

```
double y = new Rectangle().area();
sop(y);
double z = new Square().area();
sop(z);
double a = new
```


★ Reference Variable →

Reference variable is a special type of variable which is use to store the address of an object or null. means the path.

OR

It can store only two thing.

① Object Address.

② Null.

① Reference variable Declaration →

Syntax: `classType ref variable;`

Ex:-

`Sample s1;`

② Reference Variable Initialization

Syntax:-

`ref variable = Object;`

Ex:-

`s1 = new Sample();`

③ Declaration & Initialization in a single line →

Syntax:- `classType ref variable = Object;`

Ex:-

`s1 = new Sample();`

★

class Sample

```
{
    void add();
    int a = 5;
    int b = 10;
    int c = a + b;
    sop(c);
}
```

class main

```
{
    psum(-)
```

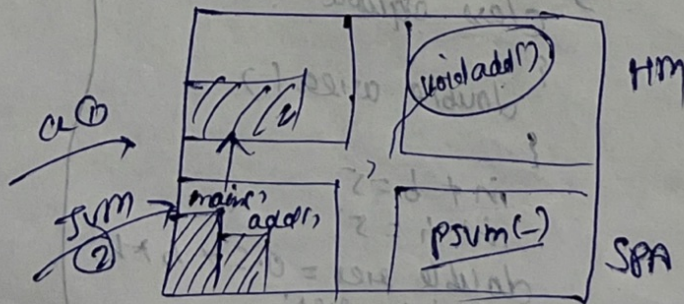
```
{
    sop("### main starts ###");
```

```
    Sample s1 = new Sample();
```

```
    s1.add();
```

```
    s1.add();
```

```
    sop("### main Ends ###");
}
```



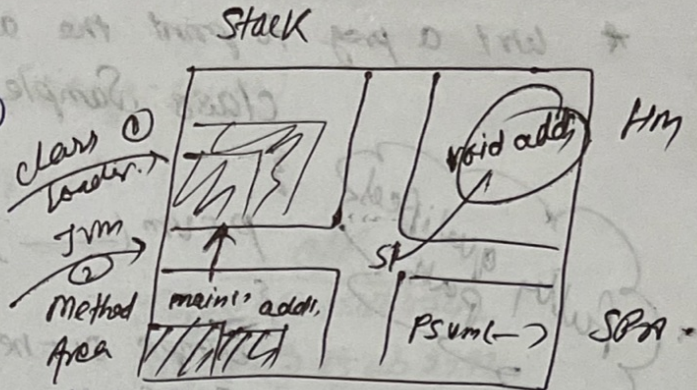
*

class Sample

```
{
    void add (inta, intb)
    {
        int c = a+b;
        System.out.println(c);
    }
}
```

class main

```
{
    public static void main (String args[])
    {
        System.out.println ("*** main start ***");
        Sample s1 = new Sample();
        s1.add(5, 5);
        s1.add(10, 5);
        System.out.println ("*** main ends ***");
    }
}
```



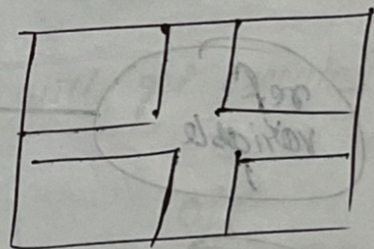
* write a prog to add 3 no. from nonstatic to static &/or the classes using reference variable with method with return type

class Sample

```
{
    int add ()
    {
        int a = 2;
        int b = 2;
        int c = 5;
        int d = a+b+c;
        return d;
    }
}
```

class main

```
{
    public static void main (String args[])
    {
        Sample s1 = new Sample();
        int x = s1.add();
        sop(x);
    }
}
```

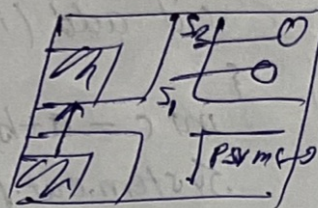


int Sample s1

★ Wrt a prog to print the address of an object.

class Sample

```
{
    psvm ( )
{
    Sample s1 = new Sample ();
    sop(s1);
}
}
```

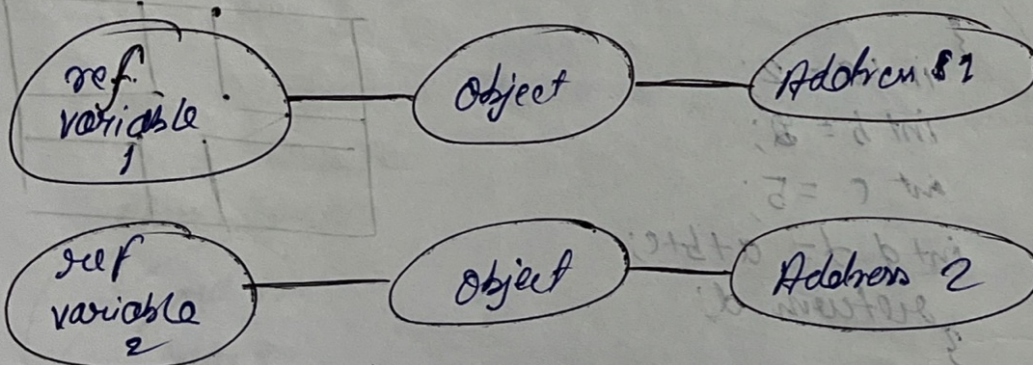


fully qualified path

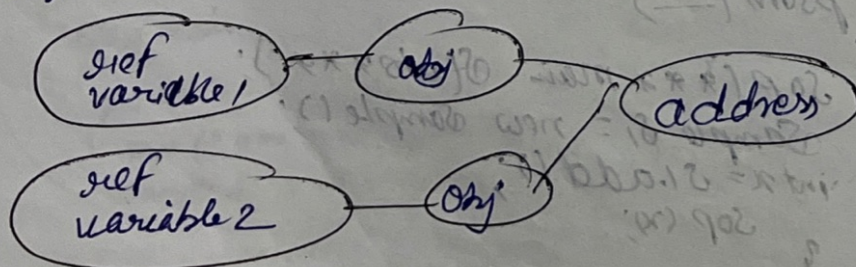
Prog package name . classname @ hexadecimal no
Sample @ 15B951

• whenever we try to print a reference variable it will give as a address of an object in fully qualified path.

★ Multiple reference variable will have multiple address for different object.



• Multiple reference variable can have same object address.




```

★ class Sample1
{
    public static void main (String[] args)
    {

```

```

        Sample1 s1 = new Sample1();
        Sample1 s2 = s1;

```

```

        sop (s1);

```

```

        sop (s2);
    }
}

```

O/P

Sample1@15db9742

Sample1@15db9742

Date → 07/10/2022

★ CONSTRUCTOR → Constructor is a special method/member, which is used to initialize the variable

Syntax:-

```

{ class name ()
{
    _
    _
}
}

```

Rules →

- (1) Constructor name should be same as class name.
- (2) Constructor will not return any value.
- (3) Constructor does not have any return type.
- (4) Constructor is always non-static.
- (5) When object is created constructor will get executed.

Ex-1 class Sample

```

{
    int x = 10;
    Sample ()
    {
        Sop ("Hey, I am a constructor");
    }
    PSvm (___)
    {
        Sop ("** Main Starts **");
        Sample s1 = new Sample ();
        Sop (s1, x);
    }
}

```

O/P

** Main Starts **

Hey, I am a Constructor

10

Date-07/10/22

Q. Wrt a prog to initialize studentid, studentname using constructor.

Ans

```

class Student
{
    int student_id;
    String student_name;
    Student (int x, String y)
    {
        student_id = x;
        student_name = y;
    }
    public static void main (String[] args)
    {
        Student s1 = new Student (1, Dingo); // Homogenous type of object creation
        System.out.println (s1.student_id);
        System.out.println (s1.student_name);
    }
}
    
```

Diagram illustrating memory allocation and object creation:

- A memory layout diagram shows a box divided into sections. One section contains 'ints id' and 'String name', with an arrow pointing to the 'Student' constructor parameters (int x, String y). Another section is labeled 'main()' and contains 'psvm'. A third section is labeled 'int x = 10'.
- An arrow labeled 'CL' points from the 'Student' class definition to the memory layout.
- An arrow labeled 'JVM' points from the 'main' method to the memory layout.
- An arrow labeled 'local to the block' points to the 'Student s1 = new Student (1, Dingo);' line.

Q. Wrt a prog to initialize subject-id or subject-name using constructor.

```

class Student1
{
    int subject_id;
    String subject_name;
    Student1 (int x, String y)
    {
        subject_id = x;
        subject_name = y;
    }
    public static void psvm (String[] args)
    {
        Student1 s1 = new Student1 (10, Dingo);
        Sop (s1.subject_id);
        Sop (s1.subject_name);
    }
}
    
```

Diagram illustrating memory allocation and object creation:

- An arrow labeled 'JVM' points from the 'psvm' method to the memory layout.
- An arrow points from the 'Student1' class definition to the memory layout.
- An arrow points from the 'Student1 s1 = new Student1 (10, Dingo);' line to the memory layout.

Date → 08/10/22

* This Keyword →

- ① This keyword is used to find out the current object.
- ② To differentiate b/w local variable and global variable if the names are same.
- ③ This keyword is always Non-static.
- ④ This keyword

Ex

class Student

```
{  
    int std-id;  
    String std-name;  
    Student (int std-id, String std-name)  
    {  
        this.std-id = std-id;  
        this.std-name = std-name;  
    }  
    public static void main (String[] args)  
    {  
        Student s1 = new Student (2, "Anil");  
        System.out.println (s1.std-id);  
        System.out.println (s1.std-name);  
    }  
}
```

Parameters are local to the block

* Composition → A class is having an object of another class is called as composition.

Class A

```
{  
    void add ()  
    {  
        int a=5;  
        int b=5;  
        int c=a+b;  
        Sop (c);  
    }  
}
```

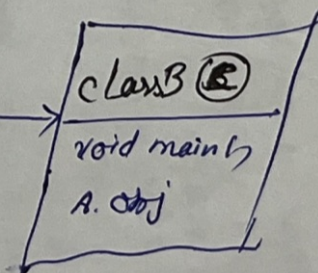
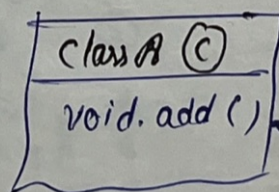
class B {

Sum (→)

A Obj = New A ();

Obj.add ();

class dig. is a pictorial representation to represent a member of the class.



Has a relationship

* BLOCKS → Java provides a separate blocks called Static Initialization Block (SIB), Instant initialization Block (IIB).

SIB $\rightarrow$ Any Block which is declared with a Keyword static is called as static initialization Block.

Syntax:-

Static }

- We can have n no. of SIB, order of execution is sequential, SIB will get execute before the main method.

ITB → Any block which is declared without a Keyword is called as ITB.

Syntax:-

3 = 2

1) we can have n-no. of IIB.

2) Order of execution is sequential.

- 2) Order of execution is sequential.
- 3) IZB will get executed, when an object is created.

class Sample

$\{ \}$

$\{ \text{Class } B \}$

* class Sample

{

Static

{

Sop("SIB1");

}

Static

Sop("SIB2");

}

psvm (—)

{

Sop("** main Starts **");

Sample s1 = new Sample();

Sop("** Main Ends **");

}

}

O/P

** Main Starts **

SIB1

SIB2

** Main Ends **

Pass By Reference →
Value

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

class Sample

{

Static void add (int a, int b)

{

int c = a + b;

Sop(c);

}

psvm (—)

{

Sop("** main Starts **");

add(5, 5);

Sop("** Main Ends **");

}

O/P

** Main Starts **

10

** Main Ends **

* Class Sample

Date - 13/10/22

```

{
    static {
        Sop("SIB");
    }
    Sample()
    {
        Sop("Hey, I am a Constructor");
    }
    {
        Sop("IIB");
    }
    psum(-2) {
        Sop("** main Starts **");
        Sample s1 = new Sample();
        Sop("*** Main Ends ***");
    }
}
    
```

O/P →

SIB

** Main Starts **

Hey, I am a constructor

IIB

** Main Ends **

Hey, I am a constructor

** Main Ends **

* Method Overloading →

Developing a multiple method with the same name but variation in the argument list is called Method Overloading.

Variation in the argument list means.

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

Rules →

- ① The method name should be same.
- ② There should be a variation in the argument list.
- ③ There is no restriction for access specifier, modifier and return type.
- ④ It can be both static and non-static.
- ⑤ We can overload main method.

Ex class whatsapp

```

{
    static void send(int x)
    {
        sop("Hi");
    }
    static void send(String y)
    {
        sop("Bye");
    }
    static void send(int x, String y)
    {
        sop("I LOVE YOU");
    }
    static void send(String y, int x)
    {
        sop("I hate You");
    }
}
class Main
{
    psvm(→)
    {
        whatsapp.send(1);
        whatsapp.send("Hello");
        whatsapp.send(1, "Hello");
        whatsapp.send("Hello", 1);
    }
}

```

dp

Hi

Bye

I LOVE YOU

I HATE YOU

* class whatsappv2

```
{
    void send (int x)
```

```
{
    sop ("Anil");
}
```

```
void send (String y)
```

```
{
    sop ("Anil Singh");
}
```

```
void send (int x, String y)
```

```
{
    sop ("I am from Jamshedpur");
}
```

```
void send (String y, int x)
```

```
{
    sop ("HOME");
}
```

```
}
```

```
class Main
```

```
{
```

```
    psum1 →
```

```
    whatsappv2 ref = new whatsappv2 ();
```

```
    ref. send (8);
```

```
    ref. send ("yes");
```

```
    ref. send (10, "No");
```

```
    ref. send ("yes", 20);
```

```
}
```

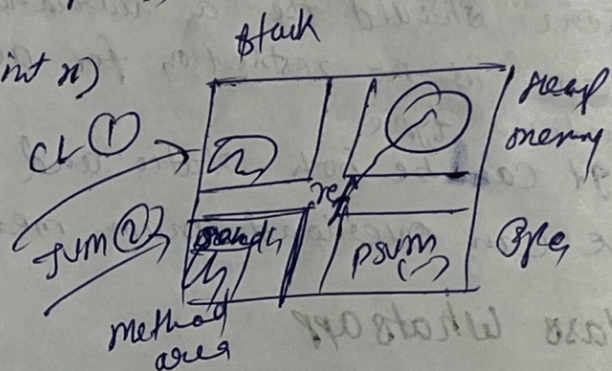
OPP

Anil

Anil Singh

I am from Jamshedpur

HOME



Date → 15/10/2022

* PASS BY REFERENCE →

Calling or invoking the method by passing the reference variable is called as Pass by Reference.

Ex.

```

class IBM_Aunt
{
    void testengg()
    {
        System.out.println("Testing 0 Testing");
    }
}

class Uncle
{
    psum(-)
    {
        IBM_AUNT a1 = new IBM_AUNT();
        Dinga.needjob(a1);
        Dinga.needjob(a1);
    }
}
    
```

class Dinga

```

{
    static void needjob (IBM_Aunt a2)
    {
        a2.testengg();
    }
}
    
```

class Dingi

```

{
    static void needjob (IBM_Aunt a3)
    {
        a3.testengg();
    }
}
    
```

Diagram illustrating Pass by Reference:

- Uncle calls psum(-) which creates IBM_AUNT a1.
- Uncle calls Dinga.needjob(a1) and Dingi.needjob(a1).
- Dinga and Dingi both receive the reference a1 and call testengg() on it.

Ex ②

```

class Java
{
    void JavaClass()
    {
        Sop("Java Developer");
    }
}

class Talah Sir
{
    psum(-)
    {
        Java J1 = new Java();
        Anil.needjava_class(J1);
        Pankaj.needjava_class(J1);
    }
}
    
```

class Anil

```

{
    static void needjava_class (Java J2)
    {
        J2.JavaClass();
    }
}
    
```

class Pankaj

```

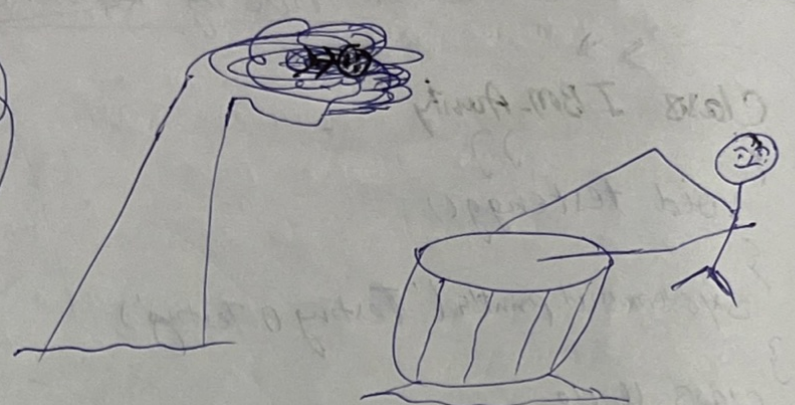
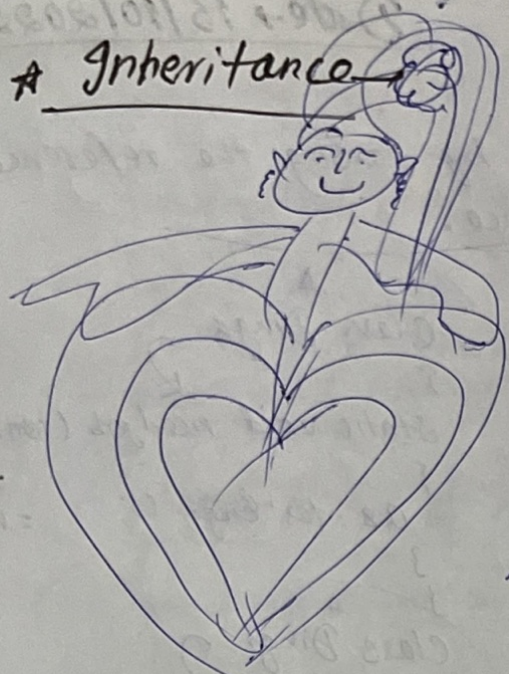
{
    static void needjava_class (Java J3)
    {
        J3.JavaClass();
    }
}
    
```

Diagram illustrating Pass by Reference:

- Talah Sir calls psum(-) which creates Java J1.
- Talah Sir calls Anil.needjava_class(J1) and Pankaj.needjava_class(J1).
- Anil and Pankaj both receive the reference J1 and call JavaClass() on it.

Date - 17/10/2022

A Inheritance

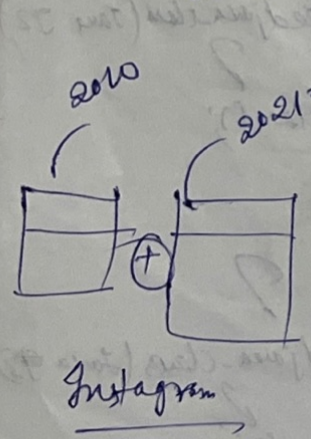


Inheriting the properties from one class to another class is called as inheritance.

In inheritance it has been classified into five types

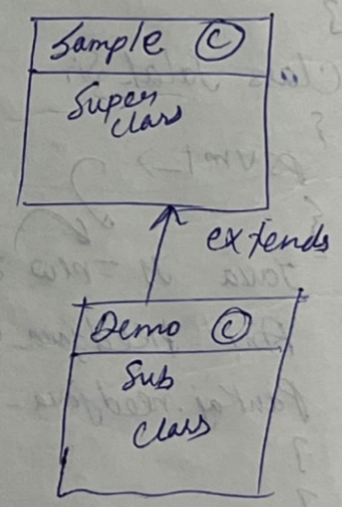
- ① Single level
- ② Multi level
- ③ Hierarchical
- ④ Multiple
- ⑤ Hybrid

① Single level Inheritance → A sub class is inheriting the properties from only one super class is called single level Inheritance



```

class Sample
{
    int x=10;
}
class Demo extends Sample
{
    void fo() //int x=10;
    {
        Sop("fofo");
    }
}
class Main
{
    psum()
    {
        Demo D1 = new Demo();
        D1.fo();
        Sop(D1.x);
    }
}
    
```



② Multilevel Inheritance →

A sub class is inheriting the properties from its super class in turn this super class inheriting the properties from one more ~~more~~ super class is called Multilevel Inheritance.

Class Sample

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

class Demo extends Sample

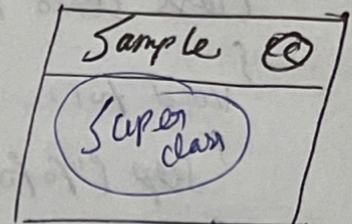
```
{
  void fo(), // int x=10;
  {
    s.op("fofo");
  }
}
```

class Memo extends Demo

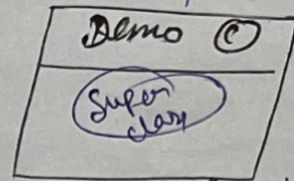
```
{
  void mo() // int x=10; // void fo()
  {
    sop("memo"); // sop("fofo");
  }
}
```

Class Main

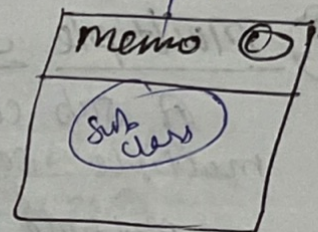
```
{
  psvm(—)
  {
    memo m1 = new Memo();
    m1.mo();
    m1.fo();
    sop(m1.m);
  }
}
```



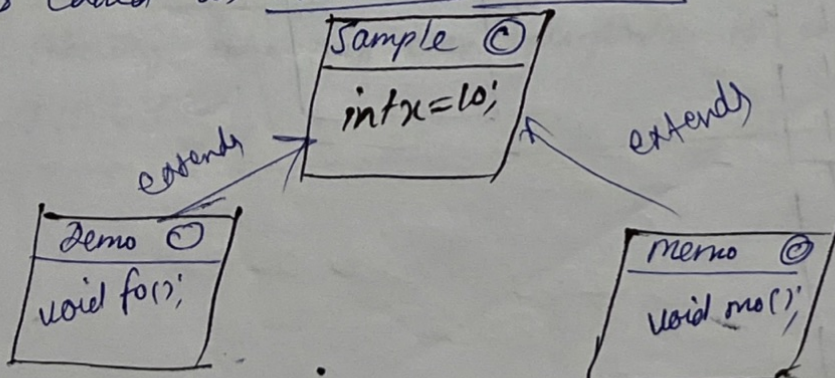
↑ extends



↑ extends



③ Hierarchical Inheritance A multiple sub class inheriting the properties from only one super class is called as Hierarchical Inheritance.




```
class Sample
```

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

```
class Demo extends Sample
```

```
{
  void fo()
  {
    sop("fofo");
  }
}
```

```
class Memo extends Sample
```

```
{
  void mo()
  {
    sop("mono");
  }
}
```

```
class Main
```

```
{
  psvm
  {
    Memo m1 = new Memo();
    m1.mo();
    sop(m1.x);
    Demo D1 = new Demo();
    D1.fo();
    sop(D1.x);
  }
}
```

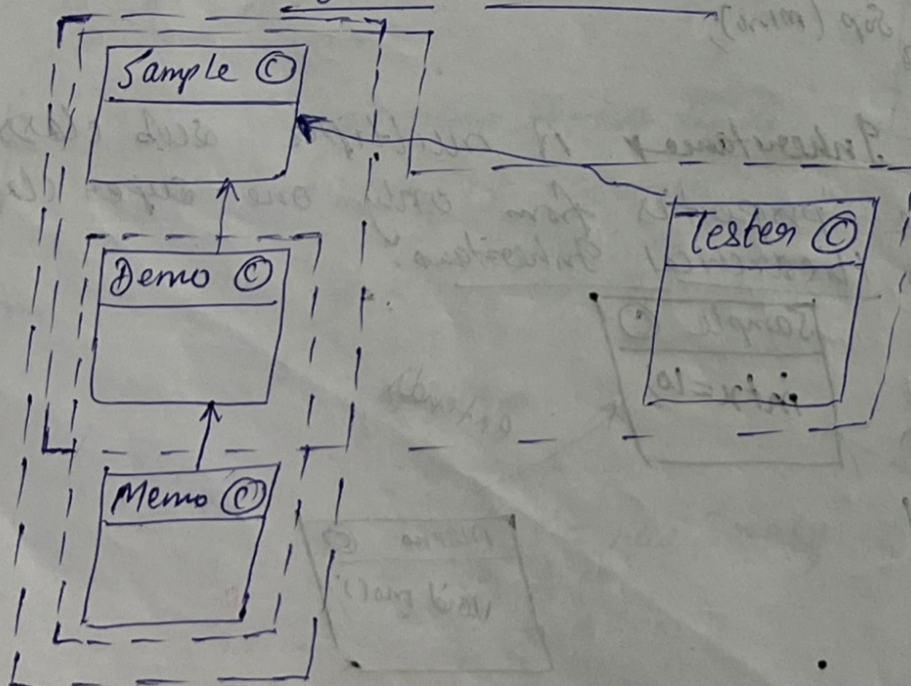
④ Multiple Inheritance →

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

Multiple Inheritance can not be achieved through classes.

It can achieve through interface.

⑤ Hybrid Inheritance → It is a combination of single level, multilevel and hierarchical is called as hybrid Inheritance.



Class Sample

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

class Demo extends Sample

```
{  
  void fo()  
  {  
    sop("fofo");  
  }  
}
```

class Memo extends Demo

```
{  
  void mo()  
  {  
    sop("momom");  
  }  
}
```

class Demo1 extends Sample

```
{  
  void do()  
  {  
    sop('dodo');  
  }  
}  
  
class Main
```

Rules → ① It is always non static.

② It should have a relationship

Advantages → Code reusability

Date- 18/10/22

Time- 11:30 AM

* Method Overriding →

Developing a method in a subclass with the same name and signature^{as} in a super class but variation different implements in the sub class is called as Method overriding.

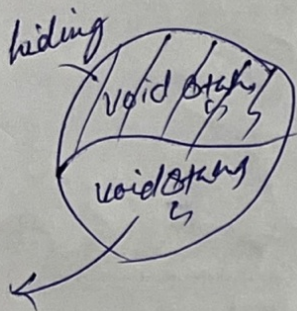
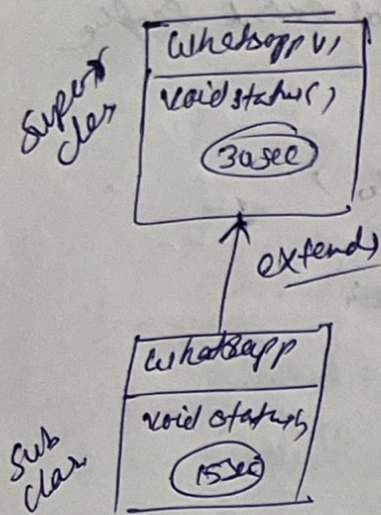
Rules →

① The method name and signature should be same.

② It should have a relationship (Inheritance)

③ It is always non-static.

* Method Overriding → Hiding the super class properties and showing only sub class properties is called Method Overriding.



Class WhatsAppV1

```

{
    void status()
    {
        sop("Status for 30 sec");
    }
}
  
```

Class WhatsAppV2 extends WhatsAppV1

```

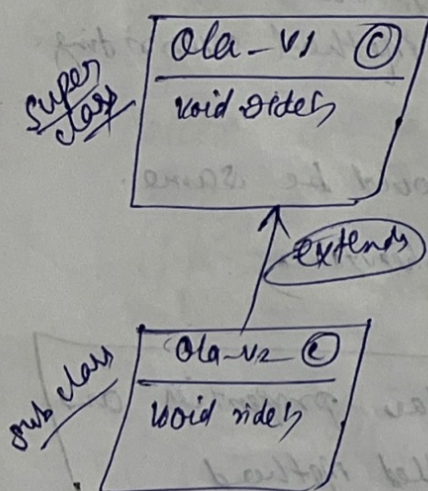
{
    void status()
    {
        sop("Status for 25 sec");
    }
}
  
```

Class main

```

{
    psvm(→)
    {
        WhatsAppV2 w2 = new WhatsAppV2();
        w2.status();
    }
}
  
```

Super Keyword → It is a keyword we use in the case of method overriding if we want both sub class implementation as well as super class implementation then we go for super keyword.



Class Ola-V1

```

{
    void ride()
    {
        sop("ride with cab!-");
    }
}
  
```

Class Ola-V2 extends Ola-V1

```

{
    void ride()
    {
        super.ride();
        sop("ride with auto bsk");
    }
}
  
```

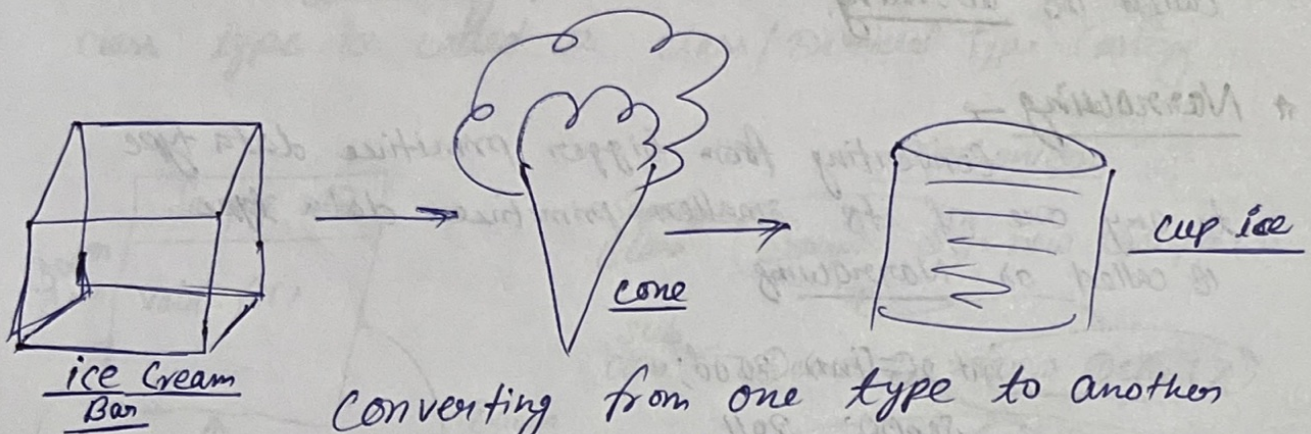
Class main

```

{
    psvm(→)
    {
        Ola-V2 o2 = new Ola-V2();
        o2.ride();
    }
}
  
```

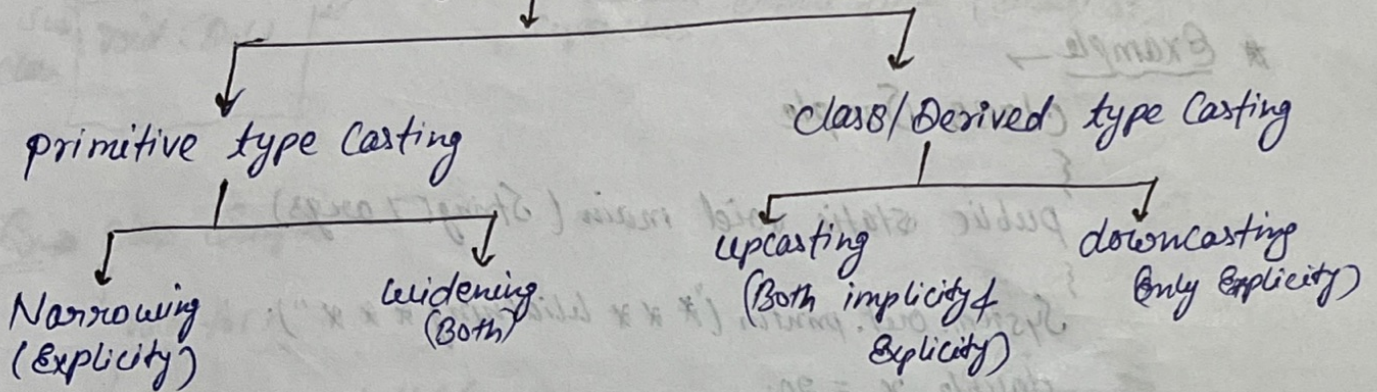

Date → 18/10/22

★ TYPE CASTING →

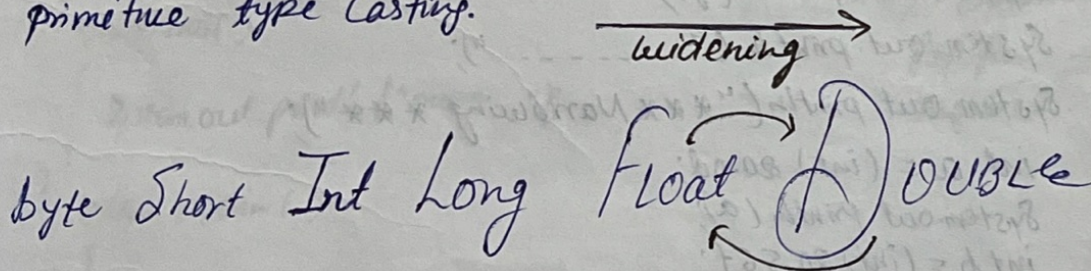


Converting from one type to another type is called as type casting.

TypeCasting



primitive Type Casting → Converting from one primitive data type to another primitive data type is called as primitive type Casting.



★ widening →
Ex:

<u>Implicitly</u>	<u>Narrowing Explicitly</u>
double x = 20; Sop(x); 20.0d	double x = (double) 20; Sop(x); 20.0d
double y = 20.56f; Sop(y); 20.56d	double y = (double) 20.56f; Sop(y); 20.56d

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

* Narrowing →

Converting from bigger primitive data type to any one of its smaller primitive data type is called as Narrowing.

```
int x = (int) 20.0d;
```

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

```
int y = (int) 20.56f;
```

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

* Example →

```
class Sample
```

```
{
```

```
    public static void main (String[] args)
```

```
{
```

```
        System.out.println ("*** Widening ***");
```

```
        double x = 20;
```

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

```
        double y = 20.56f;
```

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

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

```
        System.out.println ("*** Narrowing ***");
```

```
        int a = (int) 20.0d;
```

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

```
        int b = (int) 20.56f;
```

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

```
    }
```

O/P →

```
*** Widening ***
```

```
20.0
```

```
20.5599999465
```

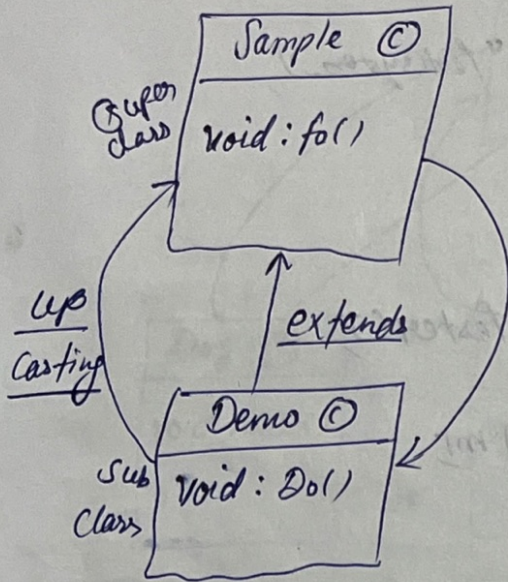
```
*** Narrowing ***
```

```
20
```

```
20
```


② Class/Derived Type Casting →

Converting from one class object to another class type is called as class/Derived Type Casting.



Mechanism

Super Class ← Sample S1 = new Sample();

Sub Class ← Demo D1 = new Demo();

Sample S1 = new Demo();

Super Class ← Demo D1 = (Demo) S1;

Ex → Class Sample

```

{
    void fo()
    {
        System.out.println("fofo...");
    }
}
class Demo extends Sample
{
    void mo()
    {
        System.out.println("momo...");
    }
}
class Main()
{
    ps vm(-);
    Sop("*** upcasting ***");
    Sample S1 = new Demo();
    S1.fo();
    Sop("*** Downcasting ***");
    Demo d1 = (Demo) S1;
    d1.fo();
    d1.mo();
}
  
```

O/P →

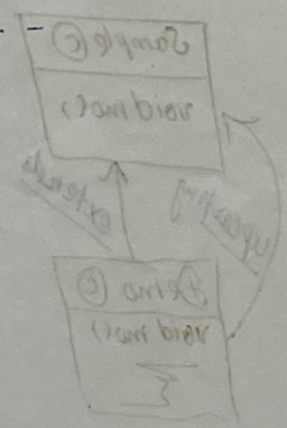
*** upcasting ***

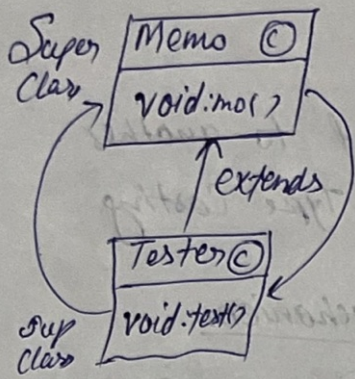
fofofo---

*** Downcasting ***

fofofo---

momo---





```

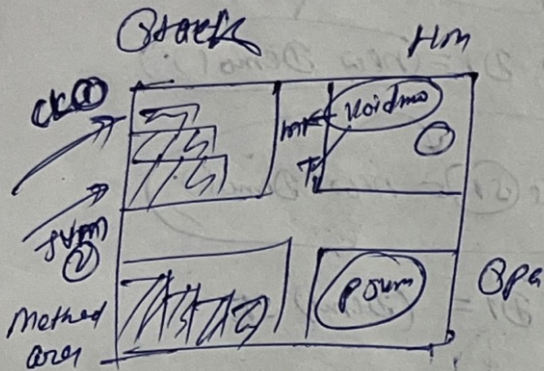
class Sample memo
{
    void mo()
    {
        System.out.println("memo");
    }
}
  
```

```

class Tester extends memo
{
    void test()
    {
        System.out.print("testSystem");
    }
}
  
```

```

class main
{
    psum()
    {
        memo m1 = new Tester();
        m1.mo();
        Tester t1 = (Tester) m1;
        t1.mo();
        t1.tex();
    }
}
  
```

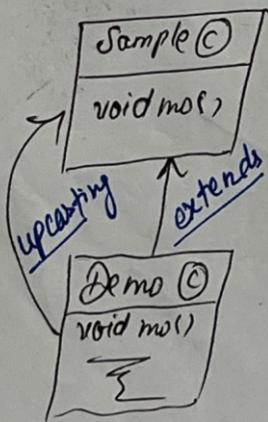


Date-22/10/22

CASE STUDY

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

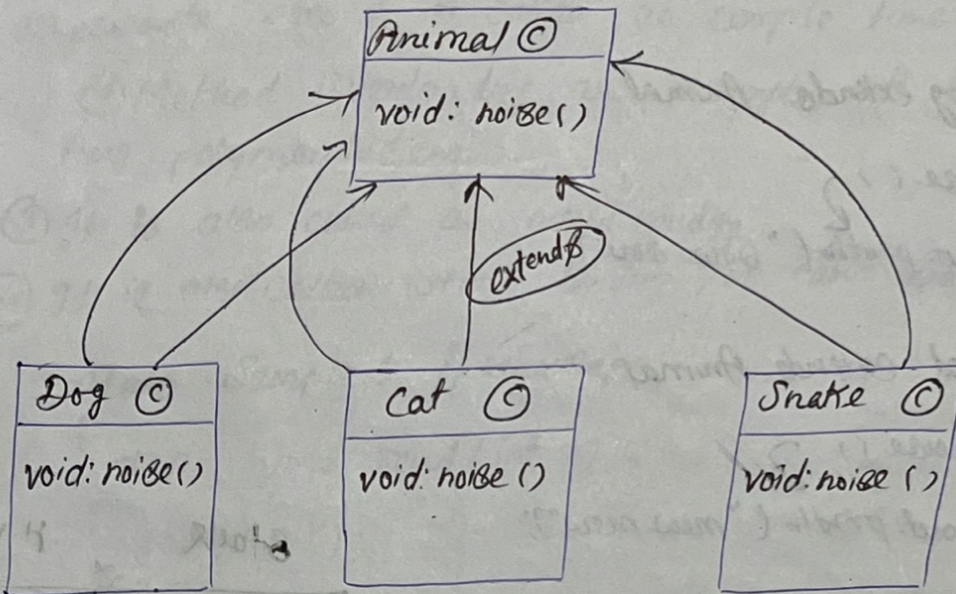
Ex-1



```

class Sample
{
    void mo()
    {
        sop("Hello");
    }
}
class Demo extends Sample
{
    void mo()
    {
        sop("Bye");
    }
}
class Main
{
    psum() {
        Sample s1 = new Demo();
        s1.mo();
    }
}
  
```


POLYMORPHISAM → An Object Showing different behaviour at different stages of its life cycle is called as polymorphisam.



Ex →

```
class Animal
```

```
{
```

```
void noise()
```

```
{
```

```
System.out.println("Some Noise");
```

```
}
```

```
}
```

```
class Dog extends Animal
```

```
{
```

```
void noise()
```

```
{
```

```
System.out.println("Bow Bow");
```

```
}
```

```
}
```

```
class Cat extends Animal
```

```
{
```

```
void noise()
```

```
{
```

```
System.out.println("mew mew");
```

```
}
```

```
}
```

```
class Snake extends Animal
```

```
{
```

```
void noise()
```

```
{
```

```
System.out.println("Buzz Buzz");
```

```
}
```

```
}
```

```
class Stimulator
```

```
{
```

```
static void ansim (Animal a1)
```

```
{
```

```
a1.noise();
```

```
}
```

```
}
```

```
class Main
```

```
{
```

```
public static void main (String[] args)
```

```
{
```

```
Dog d1 = new Dog();
```

```
Cat c1 = new Cat();
```

```
Snake s1 = new Snake();
```

```
Stimulator.ansim(d1);
```

```
Stimulator.ansim(c1);
```

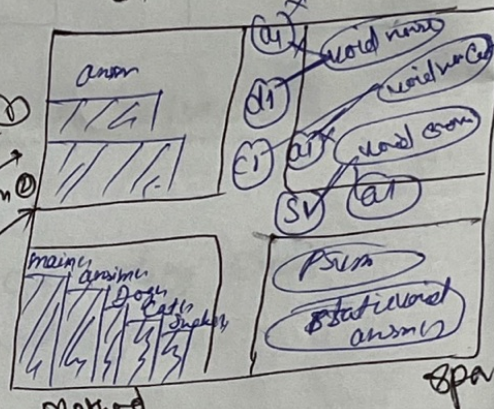
```
Stimulator.ansim(s1);
```

```
}
```

```
}
```

Stack

H.M



Method area

```
= d1  
= new Dog();  
= c1  
= new Cat();  
= s1  
= new Snake();
```

Run

COMPILE TIME POLYMORPHISM →

① The method declaration getting binded to its definition at a compile time by the compiler based on arguments passed is called as compile time polymorphism.

② Method Overloading an example of compile time polymorphism.

③ It is also called as early binding.

④ It is also called static binding, it can't be

Ex → `class Sample1 // whatsapp`

```
{  
    static void send (int x)  
    {  
        sop ("Hi");  
    }  
}
```

`class Sample2`

```
{  
    static void send (String s)  
    {  
        sop ("Hello Baby");  
    }  
    static void send (int x String s)  
    {  
        sop ("Hello Anil, How are you?");  
    }  
    static void send (String s int x)  
    {  
        sop ("I love you");  
    }  
}
```

`class Main`

```
{  
    psum (→)  
    {  
        Sample1.send (1);  
        Sample1.send ("Baby");  
        Sample1.send (1, "Baby");  
        Sample1.send ("Baby", 1);  
    }  
}
```


Run time Polymorphism →

The method declaration getting binded to its definition at the run time by the JVM whenever ~~the~~ an object is created is called as Run time Polymorphism.

Method Overriding is an example of Run time Polymorphism.

It is also called as Late binding.

It is also called dynamic binding.

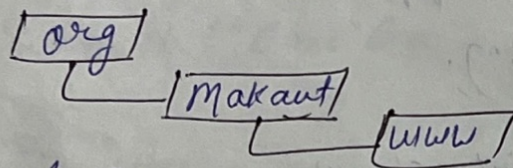
Date - 31/10/22

A PACKAGE →

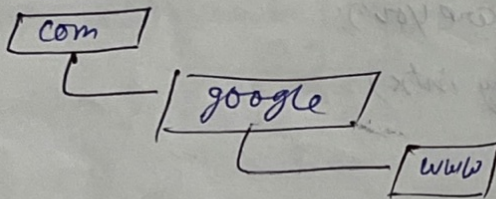
Packages are the folder structure which is used to store similar kinds of project.

Package structure should be written in the form of reverse order of the URL.

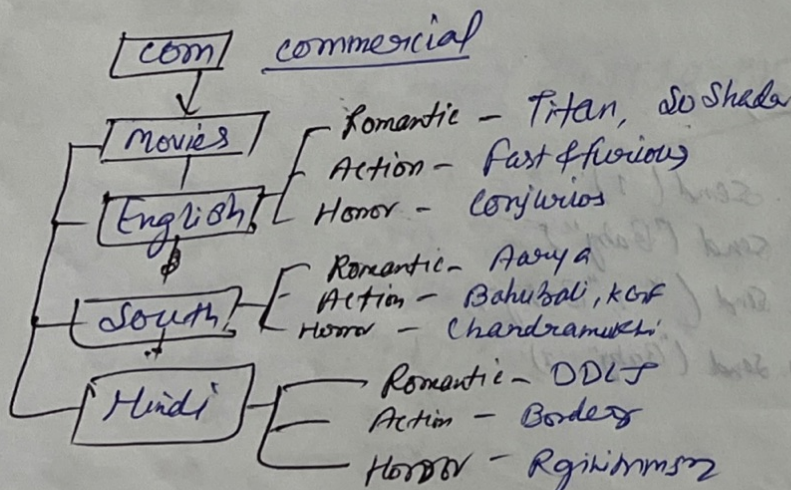
Ex ① www.Makaut.org



② www.google.com



★



④ import com.movies.english.romantic.Titanic;
import com.movies.south.action.bahubali;
import com.movies.hindi.honor.ragimms2;

★ Java provides a keyword called import to copy the files from one package to another package

- 1) In java package should be first statement.
- 2) 2nd statement should be import.
- 3) 3rd statement should be any java type (class, enum, annotation, interface)

Date - 02/11/2022

★ ABSTRACT CLASS →

Any class which is declared with a keyword called Abstract is called as Abstract class. Ex. abstract class Sample

★ Concrete class → A class which is having a concrete method is called as concrete class. Ex.

```
class Sample {
    void fo ()
    {
        //
    }
}
```

★ Concrete method → A method which is having both declaration and definition is called as concrete method.

Ex.

```
→ void fo ()
{
    //
}
```

★ ABSTRACT METHOD → A method which is declared with a keyword called Abstract is called as Abstract method. Abstract method will not have a Body

Ex.

```
Abstract void disp();
```

Rule →

① A class which is having a abstract method that class should be declared as Abstract.

② A class which provides implementation to the abstract method is called as implementation class.

Ex. abstract class Demo {

abstract void disp();

void disp ()

{ }

class Sample extends Demo

{ void disp () {

disp ("Hi");

}

- ③ It will supports constructor.
- ④ An abstract class can have both concrete method & Abstract method.
- ⑤ If any one of the abstract method is not overhiden then compulsory that class should be declare as Abstract.

Scenario 1 :-

```

abstract class Demo
{
    abstract void disp();
}
class Sample extends Demo
{
    void disp()
    {
        System.out.println("Hi");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Sample s1 = new Sample();
        s1.disp();
    }
}

```

O/P
Hi

Scenario 2 :-

```

abstract class Demo
{
    abstract void disp();
    abstract void fo();
}
abstract class Sample extends Demo
{
    void disp()
    {
        System.out.println("Hi");
    }
}
class Demo extends Sample
{
    void fo()
    {
        System.out.println("fofo");
    }
}
class Main
{
    public static void main(String[] args)
    {
        Memo m1 = new Memo();
        m1.disp();
        m1.fo();
    }
}

```

O/P
Hi
fofo

★ BLOCK → Java provides a separate block called SIB and IIB.

SIB stands for Static Initialization Block.

IIB stands for Instance Initialization Block.

SIB → Any block which is declared with a keyword Static is called as Static Initialization Block.

Syntax:-

Static {

}

- ① we can have n no of SIB, then
- ② The order of execution is Sequential
- ③ SIB will get executed before the main method.

Ex. class main

```
{  
    static {  
        System.out.println("SIB1");  
    }  
    static {  
        sop("SIB2");  
    }  
    psum(→  
    {  
        sop("xx Main Starts xx");  
        sop("xx Main Ends xx");  
    }  
    static {  
        sop("SIB3");  
    }  
}
```

O/P

SIB1

SIB2

SIB3

xx Main Starts xx

xx Main Ends xx

IIB → Any block which is declared without a keyword called Static is called as Instance Initialization Block.

Syntax:-

{

}

- ① we can have n no of IIB.
- ② The order of execution is sequential
- ③ IIB will get executed whenever object is created.


```

Q. class main
{
    sop("ITB1");
}
{
    sop("ITB2");
}
psvm(-)
{
    sop("**main starts**");
    new main();
    sop("**main ends**");
}
}

```

O/P

```

** main starts **
ITB1
ITB2
** main ends **

```

Date → 07/11/2022

* INTERFACE →

Q → when to go for interface?

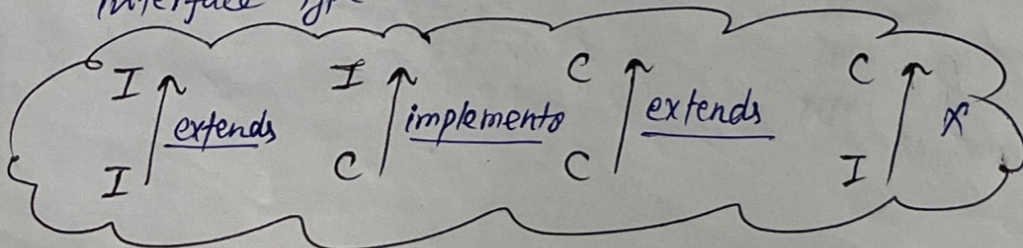
Ans → when we know 0% of implementation.

Q. when we go for Abstract?

Ans → when we know 50% of implementation.
(partially)

• INTERFACE →

- ① Interface is a java type.
- ② It is a pure abstract body.
- ③ In Interface we have only two members.
 - variable.
 - Methods.
- ④ Interface does not support constructors.
- ⑤ Variable is by default static & final.
- ⑥ Method is by default public & abstract.
- ⑦ Java provides a keyword called extends to inherit the property to interface to interface.
- ⑧ Java provides a keyword called implements to inherit a property from interface to class.
- ⑨ we cannot create object for Interface and Abstract class.
- ⑩ But we can create a reference variable of the interface type.



- (11) Through Interface we can achieve 100% abstraction.
- (12) Interface is a Supermost type in Java, it will not extend from object class.
- (13) Each and every class extends object class

Ex → ① interface Sample

```

{
    void fo();
}
class Demo implements Sample
{
    public void fo()
    {
        System.out.println("Fofo");
    }
}
public class Main
{
    public static void main(String[] args)
    {
        Demo d1 = new Demo();
        d1.fo();
    }
}

```

O/P
Fofo

Ex ② interface Nike

```

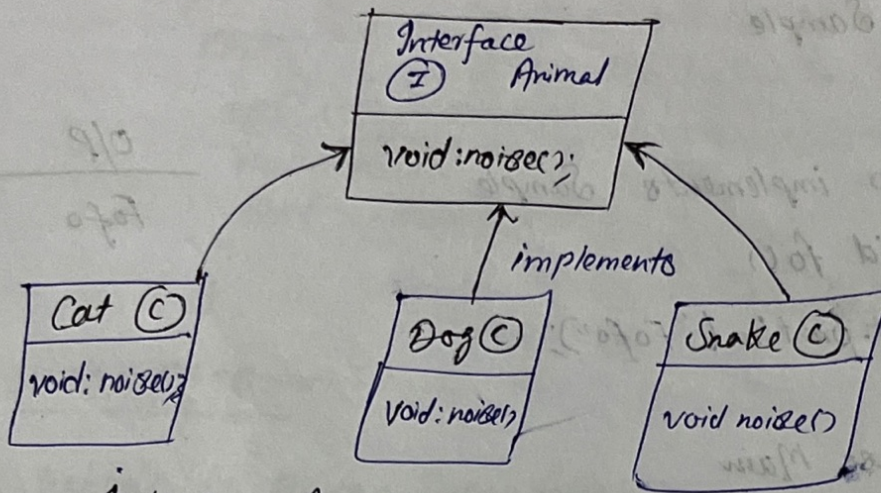
{
    void shoes();
}
interface Puma extends Nike
{
    void bags();
}
abstract class Hub implements Puma
{
    public void shoes()
    {
        System.out.println("Jingalaala Shoes");
    }
}
class EKart extends Hub {
    public void bags() {
        sop("Super Bags");
    }
}
public class Rajoutlet {
    public static void main(String[] args) {
        EKart E1 = new EKart();
        E1.bags();
        E1.shoes();
    }
}

```

O/P
Super Bags
Jingalaala Shoes

Date → 08/11/2022

* ABSTRACTION → Hiding the complexity of the system and exposing the required functionalities to the end user is called as Abstraction.



Ex1

```

Interface Animal
{
    void noise();
}

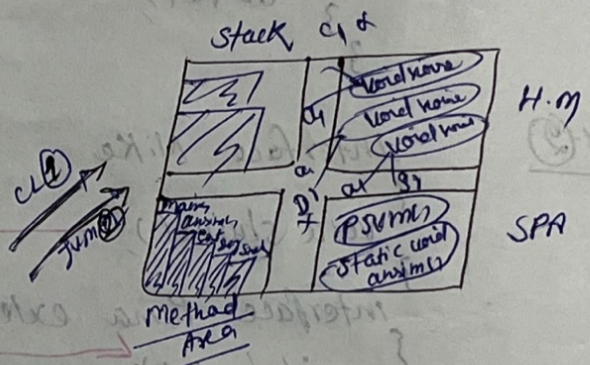
class Cat implements Animal
{
    public void noise()
    {
        sop("Mew Mew");
    }
}

class Dog implements Animal
{
    public void noise()
    {
        sop("Bow Bow");
    }
}

class Snake implements Animal
{
    public void noise()
    {
        sop("Buzz Buzz");
    }
}

class Stimulator
{
    static void ansim(Animal a)
    {
        a.noise();
    }
}

class Main
{
    public static void main()
    {
        Cat c1 = new Cat();
        Dog d1 = new Dog();
        Snake s1 = new Snake();
        Stimulator.ansim(c1);
        Stimulator.ansim(d1);
        Stimulator.ansim(s1);
    }
}
  
```

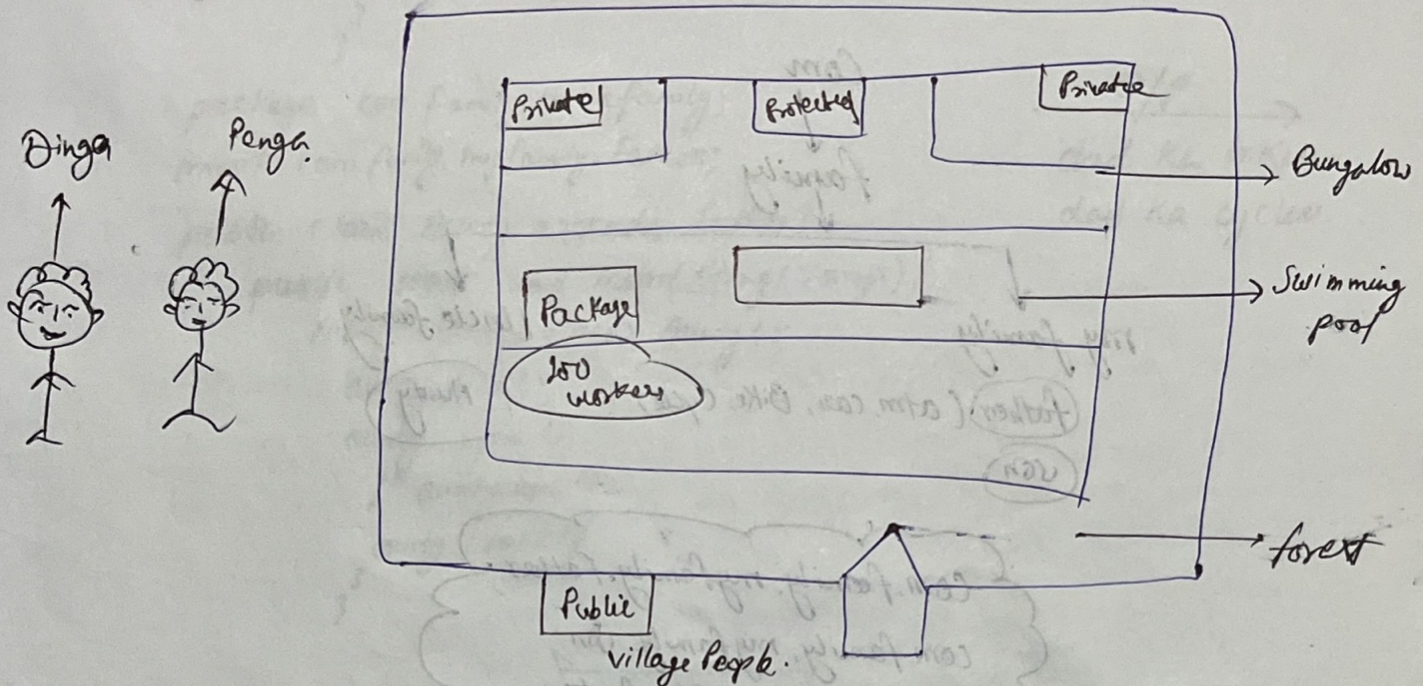


Abstraction

Polymorphism

- ① Create a reference variable of interface type.
- ② Declare all the essential properties in the interface And give the implementation in the class.
- ③ Through interface we can achieve 100% Abstraction.

★ Access Specifier →



It is used to restrict the access from one class to another class. we have four types of Access specifier.

- ① Public
- ② Protected
- ③ Private
- ④ Package or default

① Public → Any member declared with a keyword public is called as public access specifier.

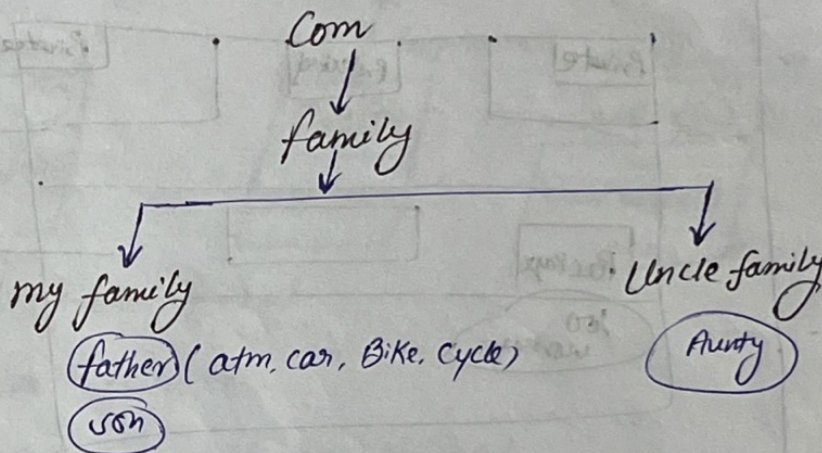
- ① It can be access within the class, outside the class, outside the package or anywhere.

② Protected → Any member which is declared with a keyword protected is called as protected access specifier.

- It can be access within the class, within the package and outside the package with a is a Relationship.

③ PRIVATE → Any member which is declared with a Key word private is called as private access specifier.
 • It can be access only within the class.

④ Package/default → Any member which is declared without any keyword is called as Package/default access specifier.
 • It can be access within the class, within the package.



com.family.myfamily.father
 com.family.myfamily.son
 com.family.myfamily.Aunt

Ex Package com.family.myfamily;
 Public class father {
 private void atm ()
 { sop ("dad Ka Atm--");
 }
 void car ()
 { sop ("dad Ka Car--");
 }
 protected void Bike ()
 { sop ("dad Ka bike--");
 }
 public void cycle ()
 { sop ("dad Ka cycle--");
 }
 }


```

package com.family.myfamily;

public class Son {

    public static void main (String[] args) {
        Father f1 = new Father();
        f1.car();
        f1.bike();
        f1.cycle();
        // f1.atm();
    }
}

```

O/P →
 dad Ka Caruu...
 dad Ka biKuu...
 dad Ka cycluu...

```

package com.family.unclefamily;
import com.family.myfamily.Father;

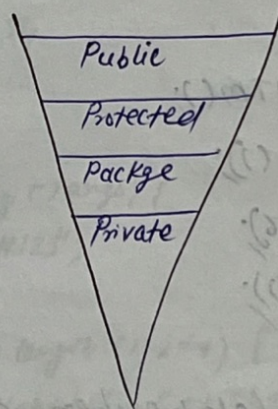
public class Aunt extends Father {
    public static void main (String[] args) {
        Aunt aumty = new Aunt();
        aumty.bike();
        aumty.cycle();
        // aumty.atm();
        aumty.car(); //
    }
}

```

O/P →
 dad Ka biKuu...
 dad Ka cycluu...

private → atm
 package → car
 protected → Bike
 public → Cycle

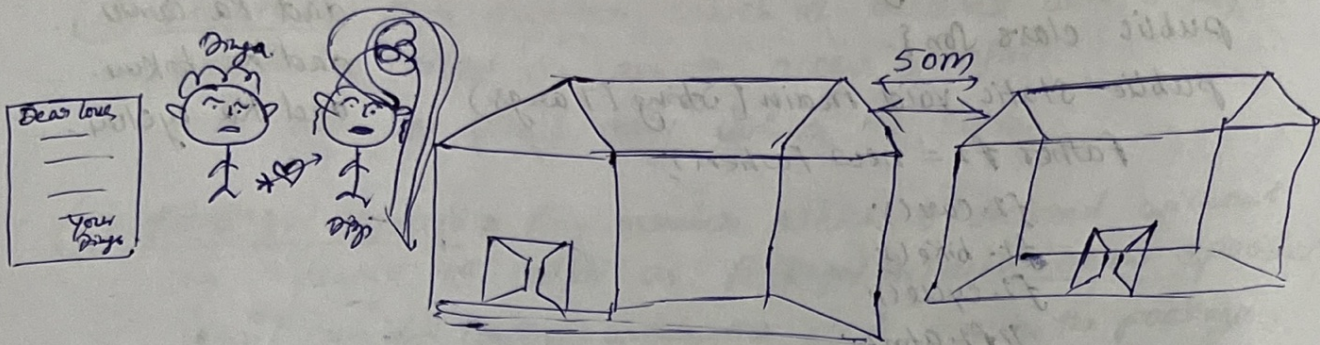
visibility ↓
 security ↑



visibility ↑
 security ↓

* Encapsulation →

- ① Java is by default Encapsulated.
 - ② It is one of the OOPS concept/principle in Java.
 - ③ We can not declare printing statement outside the method.
- Wrapping up a data members and function members into a single unit is called as Encapsulation.



* Class ATM

```
{
    private int pin = 1234;
```

```
    public void setpin (int pin)
```

```
    {
        this.pin = pin;
```

```
    }
    public int getpin()
```

```
    {
        return pin;
```

```
    }
}
```

Class Main

```
{
    psum (->
```

```
    {
        ATM a1 = new ATM();
```

```
        Sop (a1.getpin());
```

```
        a1.setpin (3456);
```

```
        sop (a1.setpin());
```

```
    }
}
```

Declare a datamember as private and restrict its direct access outside the class and provide indirect access through public services called getters & setters is called as Encapsulation.

It is also called Java Bean class.

(Object)

★ class Irctc

```

{
    private int password = 89678;
    public void setpassword (int password)
    {
        this.password = password;
    }
    public int getpassword ()
    {
        return password;
    }
}

```

```

class Main
{
    public static void main (String[] args)
    {
        Irctc I1 = new Irctc ();
        Sop (I1.getpassword());
        I1.setpassword (98754);
        Sop (I1.getpassword());
    }
}

```

O/P
89678
98754

Date →

★ Package pgm

```

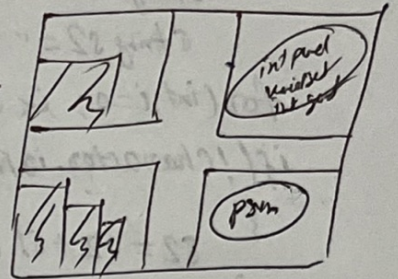
Public class Jngalala {
    public static void main (String[] args) {
        String s1 = "Java@!#123";
        int splcount = 0;
        for (int i = 0; i < s1.length (); i++) {
            if (! Character.isAlphabetic (s1.charAt (i)) & ! Character.isDigit (s1.charAt (i)))
            {
                splcount++;
            }
        }
        sop (splcount);
    }
}

```

NOTE → ① character is a class which belongs to java.lang package.

② isAlphabetic is a static method

③ isDigit is a static method.



★ Package Pgm;

```
public class Jngalala {
    public static void main (String[] args) {
        String s1 = "Java@!#123";
        String s2 = "";
        for (int i=0; i < s1.length(); i++) {
            if (!Character.isAlphabetic (s1.charAt(i)) && !Character.isDigit (s1.charAt(i)))
            {
                s2 = s2 + s1.charAt(i);
            }
        }
        System.out.println (s2);
    }
}
```

★ package pgm;

```
public class Jngalala {
    public static void main (String[] args) {
        String s1 = "Java@!#123";
        String s2 = "";
        for (int i=0; i < s1.length(); i++) {
            if (Character.isDigit (s1.charAt(i)))
            {
                s2 = s2 + s1.charAt(i);
            }
        }
        System.out.println (s2);
    }
}
```

O/P
123

★ Wrt a pg. to fetch only special character & Alphabet

Package Pgm;

```
public class Jngalala {
    psum (→) {
        String s1 = "Java@!#123";
        String s2 = "";
        for (int i = 0; i < s1.length(); i++) {
            if (!Character.isDigit (s1.charAt(i)))
            {
                s2 = s2 + s1.charAt(i);
            }
        }
        Sop (s2);
    }
}
```

O/P
Java@!#

* Wrt a prg to count. of space

package lpm;

public class Jngala {

psvm (->)

String s1 = "J a v a @ ! # 1 2 3";

int whitespace = 0;

for (int i=0; i < s1.length(); i++) {

if (Character.isWhitespace(s1.charAt(i)))

{ whitespace++;

}

System.out.println(whitespace);

}

"143 @ B H L S O S G S T S B T R # 143"

① Count space

② S is repeated

③ remove BPL character

④ check No. palindrom.

TASK

for (int i=0; i < s1.length(); i++)

{ if (Character.isDigit(s1.charAt(i)))

{ s2 += s1.charAt(i);

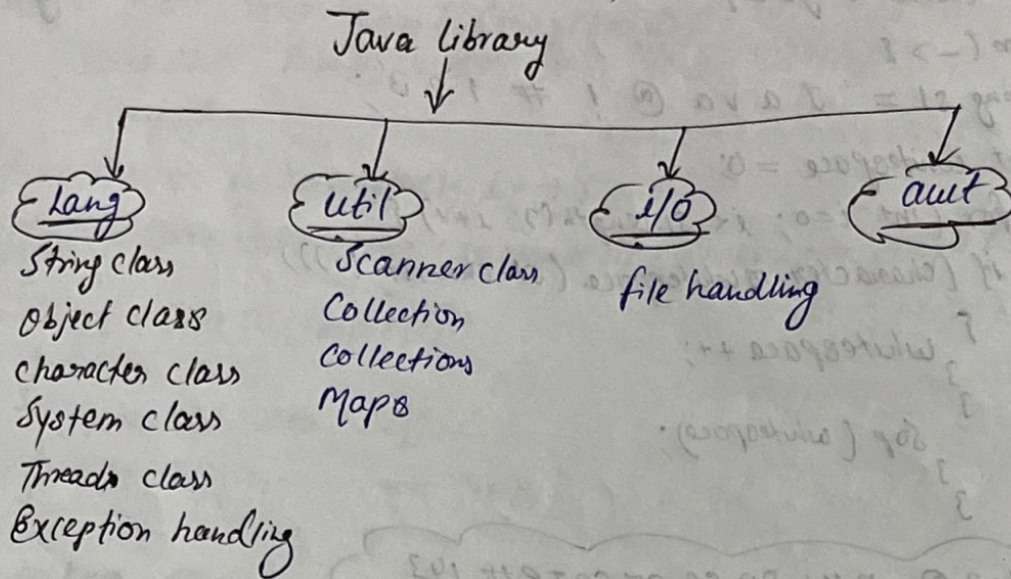
System.out.println(s2);

if (Character.isDigit(s1.charAt(i)))

Two
Ham
can
don
not
any
other
Bm

★ JAVA LIBRARY →

Java library consist of the inbuilt packages.



- ① Each and every package internally import lang package.
- ② Each and every class extends Object class.
- ③ Object class is a supermost class in a java.

★ OBJECT CLASS →

- ① Object class is a supermost class in java.
- ② It belongs to java.lang package.
- ③ Each and every class extends Object class.

★ Class Object

```

{
    public String toString()
    public int hashCode()
    public boolean equals()
    public Object clone()
    public void notify()
    public void notifyAll()
    public void wait()
    public void finalize()
}

```

Threads
Concepts

class Sample

```

{
    p sum(→
    {
        Sample s1 = new Sample()
        sop(s1);
    }
}

```


• toString() →

- ① It is a non-static method of Object class which belongs to java.lang package.
- ② It is also a Non-final method.
- ③ Whenever we print a reference variable, it will invoke toString(), and toString() will return fully qualified path in the form of String.

Syntax:-

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

- ④ We can override toString().

Ex-1 Package ObjectClassTopic;

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

O/P →

ObjectClassTopic.Sample@15d69742

Ex-2 Override toString().

package ObjectClassTopic;

public class Sample {

public String toString() {

return "HEY";

}

public static void main(String[] args) {

Sample s1 = new Sample();

System.out.println(s1);

}

O/P

HEY

• hashCode() → ① It is a Non-Static or Non-Final method which belongs to java.lang package.

- ② The return type of hashCode() is int.
- ③ Whenever we invoke a hashCode(), it will give a unique no. called hash no.

Syntax:-

```
public int hashCode()  
{  
    _____  
}  
    _____
```


Ex → package ObjectClassTopic;

public class Sample {

public static void main (String[] args)

{

Sample s1 = new Sample();

Sop (s1.hashCode());

Sample s2 = new Sample();

Sop (s2.hashCode());

}

O/P →

366712642

1829164700

Date-12/11/2022

★ Equals() → It is non-static / non-final method of an object class. which belongs to java.lang package.

② The return type of the equals () is Boolean.

③ equals () is used to compare the object ~~address~~ value.

Syntax:

```
public boolean equals()
{
    //
}
```

Ex →

package ObjectClassTopic;

public class Sample {

public static void main (String[] args)

{

Sample s1 = new Sample();

Sample s2 = new Sample();

System.out.println (s1.equals(s2));

}

/* Sample s2 = s1;

Sop (s1.equals(s2)); */

O/P →

False

O/P,

True

★ FINAL CLASS → A class which is declared with a keyword final is called as final class.

① we can not inherit the final class but we can create object for final class.

③ we can not override the final class.

④ we can overload the final class class.

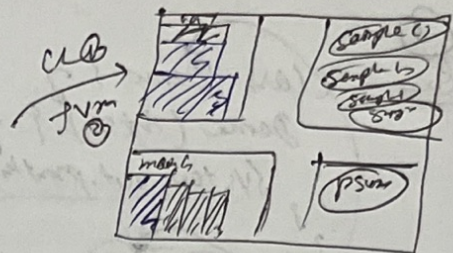
⑤ we can declare method, variable as final but not a constructor.

* CONSTRUCTOR OVERLOADING

Developing a multiple constructor with the same name but variation in argument list is called constructor overloading.

Ex,

```
public class Sample {
    Sample (int x) {
        System.out.println ("Hello");
    }
    Sample (int x, String y) {
        System.out.println ("Hey");
    }
    Sample (String y, int x) {
        System.out.println ("Bye");
    }
    public static void main (String [] args) {
        new Sample (10);
        new Sample ("Hi");
        new Sample (10, "Hi");
        new Sample ("Hi", 10);
    }
}
```



O/P →
Hi
Hello
Hey
Bye

* CONSTRUCTOR CHAINING

Constructor chaining is used to call from one constructor to another constructor.

- this () calling statement →

① this () calling statement is used to call from one constructor to another immediate constructor within a class.

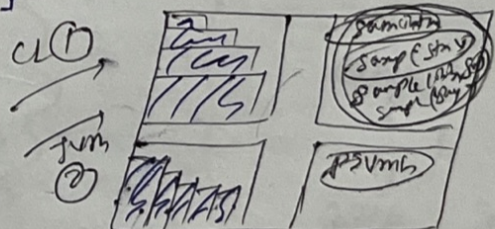
② this () calling statement should be first statement inside the constructor.

③ We cannot have multiple this () calling statement in single constructor.

Ex → package ObjectClassTopic;

```
public class Sample {
    Sample (int x) {
        Sop ("Hi");
    }
    Sample (String y) {
        this (10);
        Sop ("Hello");
    }
    Sample (int x, String y) {
        this ("Hello");
        Sop ("Hey");
    }
    Sample (String y, int x) {
        this (10, "Hello");
        Sop ("Bye");
    }
}
```

```
sum → prum ( ) {
    new Sample ("Hi", 10);
}
```



O/P →
Hi
Hello
Hey
Bye

A Super Calling Statement - Super()

① Super calling statement is used to call from one constructor to another constructor b/w the classes.

② Super Calling Statement should be the first statement

③ Ex

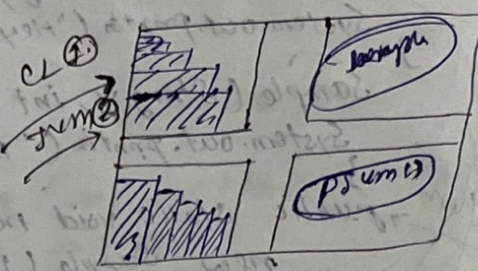
```
class Demo {
    Demo (int x) {
        System.out.println("Hi");
    }
}
```

```
class Memo extends Demo {
    Memo (String y) {
        super(10);
        Sop("Hello");
    }
}
```

```
class Nemo extends Memo {
    Nemo (int x, String y) {
        super("Hello");
        Sop("Hey");
    }
}
```

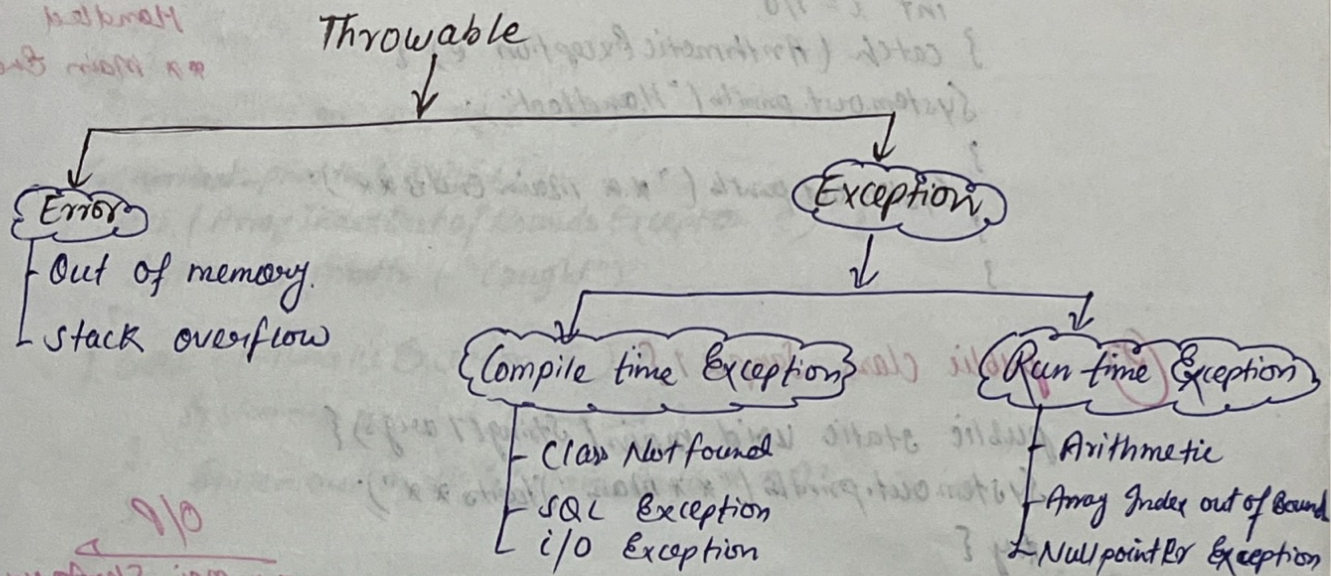
```
class Sample extends Nemo {
    Sample (String y, int x) {
        super(10, "Hello");
        Sop("Bye");
    }
}
```

```
public class Main {
    public static void main (String[] args) {
        new Sample ("Hi", 10);
    }
}
```



O/P
Hi
Hello
Hey
Bye

Exception Handling → It is an event triggered during the execution of the program which stops the execution suddenly or abruptly. Handling this event using try, catch or throws is called as Exception Handling.



- Any abnormal statement occurs it should be in a try block and it is handled by catch block.
- We can't develop any statement b/w try and catch block.
- For one try block, one catch should be there.
- One try block can have multiple catch blocks.
- It can have multiple catch blocks but one catch block will be getting addressed.
- Once the exception occurs immediately it will go to the catch block.

Syntax -

① try {

}
catch ()
{
}
}

② try {

}
catch ()
{
}
catch ()
{
}
}

③ try {

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

④ try {

}
catch ()
{
}
finally {
}
}

⑤ try {

}
finally {
}
}

Ex-① package ExceptionHandlingTopic;

```
public class Sample {
```

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

```
        System.out.println ("** Main Starts **");
```

```
        try {
```

```
            int i = 1/0;
```

```
        } catch (ArithmeticException e) {
```

```
            System.out.println ("Handled");
```

```
        }
```

```
        System.out.println ("** Main Ends **");
```

```
    }
```

```
}
```

O/P
** Main Starts **

Handled

** Main Ends **

② public class Sample1 {

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

```
        System.out.println ("** Main Starts **");
```

```
        try {
```

```
            int i = 1/0;
```

```
        } catch (ArithmeticException e) {
```

```
            System.out.println ("Handled");
```

```
        }
```

```
        catch (ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println ("Caught");
```

```
        }
```

```
        System.out.println ("** Main Ends **");
```

```
    }
```

```
}
```

O/P

** Main Starts **

Handled

** Main Ends **

③ public class Sample2 {

```
    public static void main (String[] args)
```

```
    {
```

```
        try {
```

```
            int i = 12/3;
```

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

```
            System.out.println (arr[i]);
```

```
        } catch (ArithmeticException e) {
```

```
            System.out.println ("Handled");
```

```
        } catch (ArrayIndexOutOfBoundsException e) {
```

```
            sop ("Caught");
```

```
        } sop ("** Main Ends **");
```

```
    }
```

```
}
```

O/P

** Main Starts **

Caught

** Main Ends **

④ public class Sample3 {

public static void main (String[] args) {

System.out.println ("** Main Starts **");

int a=8;

int b=0;

try {

int i = a/b;

try {

int[] arr = {90, 80, 70, 60, 50};

System.out.println (arr[i]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println ("Caught");

}

} catch (ArithmeticException e) {

System.out.println ("Handled");

}

System.out.println ("** Main Ends **");

}

}

- finally Block is a mandatory Block even though if the Exception occurs or not it will be execute mandatorily.

⑤ public class Sample4 {

public static void main (String[] args)

{ System.out.println ("** Main Starts **");

try {

int i = 10/2;

} catch (ArithmeticException e) {

System.out.println ("Handled");

} finally {

System.out.println ("Close the server");

} System.out.println ("** Main Ends **");

}

}

⑥ public class Sample5 {

psvm (-) {

System.out.println ("** Main Starts **");

try {

~~int~~ int i = 10/2;

} finally {

System.out.println ("Close the server");

} System.out.println ("** Main Ends **");

}

}

O/P
→
** Main Starts **
Handled
** Main Ends **

O/P
→
** Main Starts **
Close the server
** Main Ends **

O/P
→
** Main Starts **
Close the server
** Main Ends **

Throw → ① Throw is a Keyword which is used to create an object for Exception. ④

② Throw can create only one object at a time.

③ Throw will be in a method body.

Throws → ② Throws is used to propagate the Exception.

① It can propagate more than one exception.

③ Throws will be there in method declaration.

Ex class Sample

```
{
    psvm (___)
    {
        Sop ("** Main Starts **");
        try {
            int i = 1/0; // throw new Arithmetic Exception();
        } catch (Exception e) {
            Sop ("Handled");
        }
        Sop ("** Main Ends **");
    }
}
```

an
★ STACK UNWINDING → When exception occurs there is no one to handle the exception and at last it will propagate the main method, even in main method there is no one to handle main method, the stack will get destroyed, it is called Stack Unwinding.

Ex public class Sample {
 static void disp1() {
 int i = 1/0; }
 static void disp2() {
 disp1();
 }
 static void disp3() {
 disp2();
 }
 static void disp4() {
 disp3();
 }
}

```
public static void main (String args)
{
    Sop ("** Main Starts **");
    disp1();
    Sop ("** Main Ends **");
}
```


★ PRINT STACK TRACE () →

It is a Non-Static method of Throwable class. which will be used to find the root cause of the defects by using System.err.

Ex- package ExceptionHandlingTopic;

public class Sample {

static void disp1() {

int i=10;

} static void disp2() {

disp1();

} static void disp3() {

disp2();

} static void disp4() {

disp3();

} public static void main (String args[]) {

System.out.println("Main Starts");

try {

disp4();

} catch (Exception e) {

e.printStackTrace();

} System.out.println("Main Ends");

}

O/P

*** Main Starts ***

java.lang.ArithmeticException: / by zero

at ExceptionHandlingTopic.Sample.disp1 (Sample.java:6)

disp2 (Sample.java:10)

disp3 (Sample.java:14)

disp4 (Sample.java:18)

main (Sample.java:25)

*** Main Ends ***

* Checked Exception → ① It is also called as compile time Exception or user defined Exception.

② When an Exception occurs at a compile time and it's get caught by the compiler is called as compiled time Exception.

③ Here we use both try and catch/throws.

Ex:

class ShaadiDotCom

{ static void submit() throws ShaadiException

{ int age = 15;

if (age >= 21)

{ System.out.println("Happy Married Life");

else

{ throw new ShaadiException("Not Eligible");

① try { try {

submit();

} catch (ShaadiException e)

{ sop(e.getMessage());

}

class ShaadiException extends Exception

{ String msg;

ShaadiException (String msg)

{ this.msg = msg;

} public String getMessage()

{ return msg;

}

O/P

Not Eligible

★ Unchecked Exception →

- ① Unchecked Exception is also called as Runtime Exception.
- ② When an exception occurs at a runtime and it is handled by the JVM is called as Runtime Exception.

By,

public class Sample {

public static void main (String [] args)

{
System.out.println("** Main Starts **");

String s1 = null;

try {

System.out.println(s1.hashCode());

} catch (NullPointerException e) {

System.out.println("Handled");

System.out.println("** Main Ends **");

}
}
}

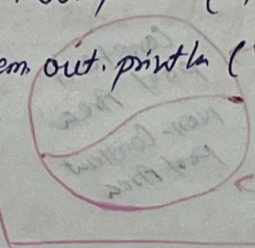
O/P

** Main Starts **

Handled

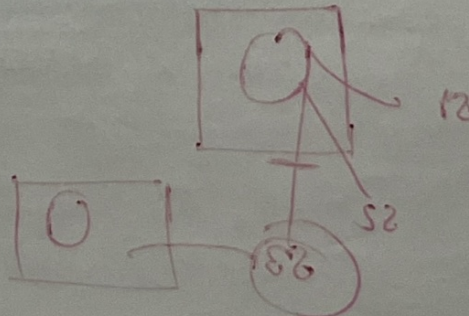
** Main Ends **

String
loop
area
}



- ① The object which is created without any new operator can get store in constant pool area.
- ② The object which is created with new operator will get stored in non-constant pool area.
- ③ The object which is created without any new operator will not allow duplicate.
- ④ The object which is created with new operator will allow duplicate.

Is String class is Immutable?
If any one of the reference variable is dereferenced, it will not affect state of an object. When multiple reference variable will be pointing to the same object address.



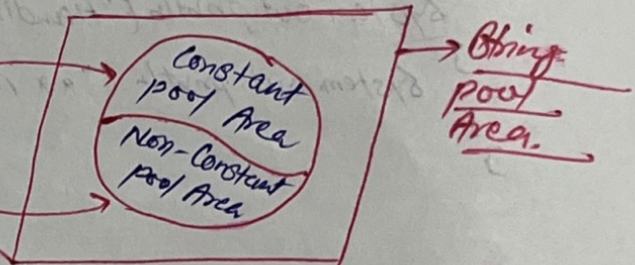
Date: 14/11/2022

* String class →

- ① String class is a final class.
- ② We can create string object which will be store in string pool Area. inside heap memory.
- ③ String pool Area is further classified into two -
 - ① Constant pool Area.
 - ② Non-constant pool Area.
- ④ We can create object in two ways -
 - ① without using new operator.
 - ② with using new operator.

String Object

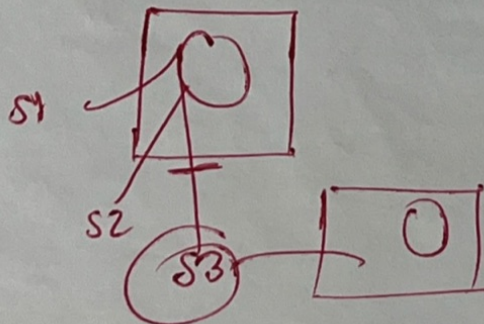
- ① without using new operator
- ② with using new operator



- ⑤ The Object which is created without using new operator will get store in constant pool Area.
- ⑥ The Object which is created with using new operator will get store in non-constant pool Area.
- ⑦ The Object which is created without using new operator will not allow duplicate
- ⑧ The Object which is created with using new operator will allow duplicate.

Q. why string class is Immutable?

Ans If any one of the reference variable is dereferenced, it will not affect state of an object, when multiple reference variable will be pointing out the same object Address.



* Difference b/w == operator and equals().

==

equals()

- ① We can not override the operator
- ② It is used to compare the Obj. Address in Object class.
- ③ It is an operator

- ① We can override the equals().
- ② It is used to compare Obj. Values in String class.
- ③ It is a Nonstatic method

Ex

package StringClassTopic;

public class Sample {

psvm(-) {

String s1 = "Hi";

// Constant pool Area > No Duplicates

String s2 = "Hi";

String s3 = new String("Hello");

// Non-constant pool Area > Duplicates

String s4 = new String("Hello");

System.out.println(s1 == s2);

System.out.println(s1.equals(s2));

System.out.println(s3 == s4);

System.out.println(s3.equals(s4));

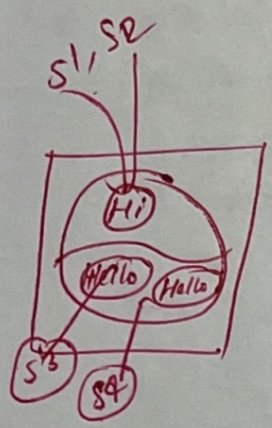
System.out.println(s1.hashCode());

System.out.println(s2.hashCode());

}

O/P

true
true
false
true
8337
2337



File class

File class

File f = new File("D:\\file handling\\Sample.txt");
if (!f.exists())
{
f.createNewFile();
}

System.out.println(f.exists());

System.out.println(f.createNewFile());

}

}

Date → 15/11/2022

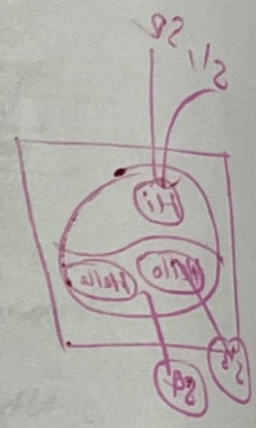
★ File Handling → Handling the files is called as file Handling.

```

class File
{
    boolean CreateNewFile()
    boolean delete()
    boolean exists()
    boolean mkdirs()
}
    
```

Ex 1 package fileHandlingTopic;
import java.io.*;

// create folder
import java.io.IOException;
public class Sample
{
 public void psvm (—) throws IOException
 {
 File f1 = new File ("D:/fileHandling");
 if (f1.mkdirs())
 {
 sop ("folder is created");
 }
 else
 {
 sop ("folder is not created");
 }
 }
}



Ex 2 Public class Sample // create file
{ Psvm (—) throws IOException {
 File f1 = new File ("D:/fileHandling/Dinje.txt");
 if (f1.createNewFile ())
 {
 sop ("file is created");
 }
 else
 {
 sop ("file is not created");
 }
}

★ package fileHandlingTopic; ★

import java.io.*;
import java.io.IOException;

public class Sample {

psvm(-) throws IOException {

File f1 = new File("D:/FileHandling/Dinga.txt");

if (f1.exists()) { // if (f1.delete())

sop("file is ~~created~~"); // sop("file is deleted");

} else {

sop("file is not exist"); // sop("file is not deleted");

}

★ public class Sample {

psvm(-) throws IOException {

File f1 = new File("D:/FileHandling/Dinga.txt");

FileWriter fw = new FileWriter(f1);

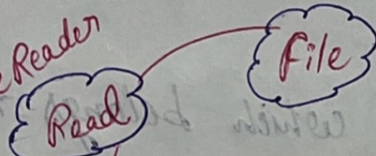
fw.write("Hello");

fw.flush();

sop("Data is written");

}

File Reader



write

FileWriter

write()

folder

Any file

createNewFile()

★ public class Sample {

psvm(-) throws IOException {

File f1 = new File("D:/FileHandling/Dinga.txt");

FileReader fr = new FileReader(f1);

char[] arr = new char[(int)f1.length()];

fr.read(arr);

sop(arr);

}

O/P

Hello

★ COLLECTION FRAMEWORKS →

PURPOSE →

To overcome the drawback of array, we go for Collection.

- ① Size is fixed.
- ② It is homogeneous.

Collection is a unified Architecture which consist of both classes and interface.

- ① In collection size is dynamic.
- ② It can store both homogeneous & heterogeneous type of data.

Collection is a interface which belongs to java.util package.

Q → How the primitive data types are getting store in collection?

Ans → In the collection the elements are store in the form of object with the help of wrapper class.

★ WRAPPER CLASS →

Wrapper class is a class which belongs to java.lang Package.

In wrapper class we have two types.

- ① BOXING.
- ② UNBOXING.

① BOXING → Converting from primitive data types to Object is called as BOXING.

② UNBOXING → Converting from Object to primitive data type is called as UNBOXING.

PRIMITIVE DATA TYPE

WRAPPER CLASS

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

Qx ★ package Collection topic;

public class sample {

prgm (-) {

sop ("*** BOXING ***");

int x = 10;

Integer i1 = new Integer(x);

sop(i1);

sop ("*** UNBOXING ***");

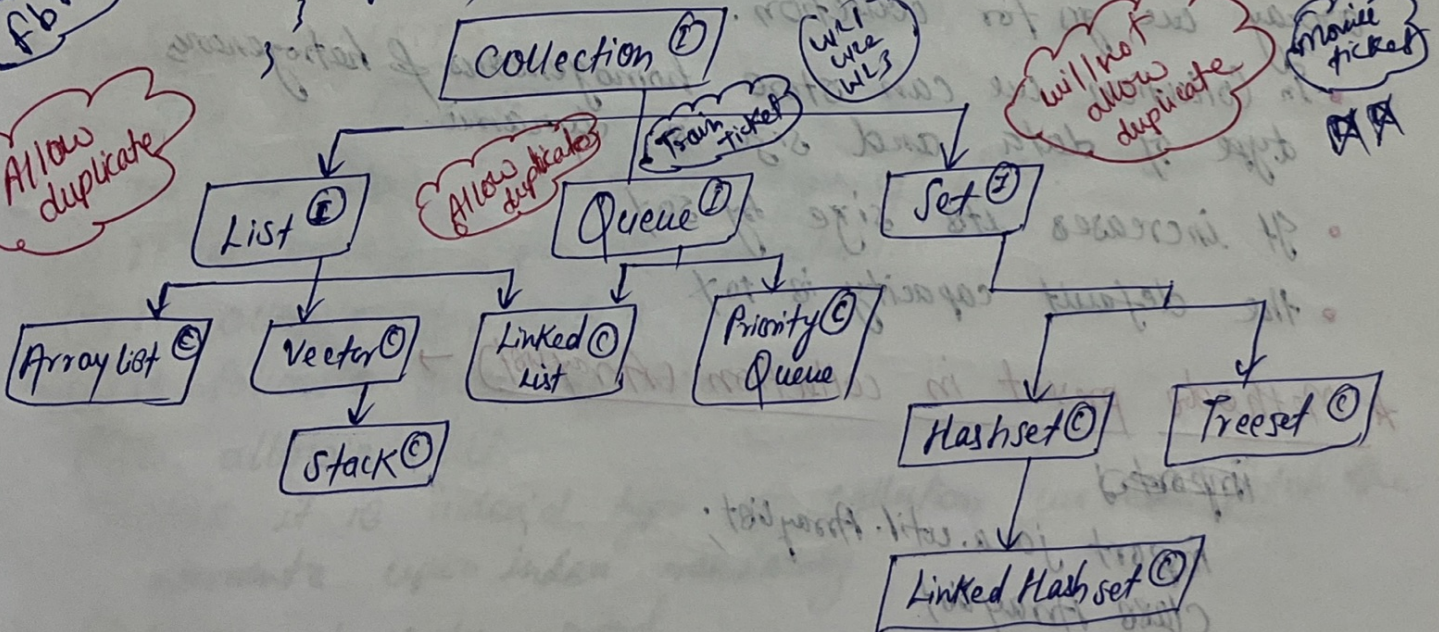
int y = (int) i1;

sop(y);

}

fb Name

Allow duplicate



05/01/23

* TYPES OF INTERFACE →

① MARKER INTERFACE →

An Empty Interface is called as Marker Interface.

Ex → cloneable

- Every collection implements cloneable interface. (marker)
- Every collection implements Serializable and

② Functional Interface →

Any Interface which is having only one abstract method in it is called functional Interface.

NOTES →

* In collection we have 3 sub interfaces

- ① List
- ② Queue
- ③ Set

* WHY COLLECTION →

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

- In Collection we can store homogeneous & heterogeneous type of data and size is dynamic.
- It increases its size by 50%.
- The default capacity is 10.

* Methods present in collection (ArrayList) →

~~import~~

import java.util.ArrayList;

class ArrayList

{
void add (Object obj) → used to add/append the element into collection.

`void add (int index, object obj)` → Used to add/append the element into collection based on the specified index.

`void addAll (collection c)` → Used to add/append one collection elements with another specified collection.

`void addAll (int index, collection c)` → Used to add/append one collection elements with another specified collection based on the specified index.

`void remove()`

`void removeAll()`

`void retainAll()`

`void size()`

`void get (int index)`

`void contains()`

}

★ LIST →

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 heterogeneous type of data.
- ③ It is index'd type of collection.
- ④ It allows duplicates.
- ⑤ It follows order of insertion.
- ⑥ It allows null.
- ⑦ Since it is index'd type of collection we can fetch the elements upon index randomly.
- ⑧ It is not auto sorted.

* List has three sub class's.

① ArrayList

② Vector

③ LinkedList

Date - 07/01/2023

① ARRAYLIST →

Features of ArrayList :-

① Size is dynamic.

② We can store heterogeneous type of data.

③ It is indexed type of collection.

④ It allows duplicates.

⑤ It 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%.

⑩ It is not synchronized.

⑪ Since it's not synchronized the performance is fast.

NOTE → If the return type is accept from ~~void~~ void, we can write in a printing statement.

$$\begin{aligned}\text{Capacity} &= \text{Current Capacity} \times \frac{3}{2} + 1 \\ &= 10 \times \frac{3}{2} + 1 \\ &= 16\end{aligned}$$

* package Collection topic;

import java.util.ArrayList;

public class Sample {

public static void main (String [] args) {

ArrayList l1 = new ArrayList();

l1.add ("Hello");

l1.add (123);

l1.add (true);

l1.add ('A');

l1.add (12.5);

l1.add (null);

l1.add ("Hello");

l1.add (null);

System.out.println (l1);

}
}

O/P →

[Hello, 123, true, A, 12.5, null, Hello, null]

② VECTOR → Vector is a class which belongs to List interface.

features of Vector:-

- ① Size is dynamic.
- ② We can store heterogeneous type of data.
- ③ It is index'd type of collection.
- ④ It allow's duplicates.
- ⑤ It follow's order of insertion.
- ⑥ It allow's null.
- ⑦ Since it is index'd type of collection we can fetch the elements upon index randomly.
- ⑧ It is not auto sorted.
- ⑨ It increases its size by 100%.
- ⑩ It is synchronized.
- ⑪ Since its synchronized the performance is slow.

③ LINKED LIST →

It is a class which has dual property.

LinkedList inherits the properties from both list and queue interface. So, we can use `get (int index)`..

Along with that we can use 2 other methods of Queue interfaces i.e:

1) peek() → It will retrieve / fetch the topmost element of the list but does not remove it.

2) poll() → It will retrieve / fetch the topmost element of the list removes it, thereby the size gets decreased.

Whenever we need proper order of insertion then we should go for LinkedList, it has dual properties i.e. It inherits the properties from both list and Queue.

★ Features of LinkedList →

- ① Size is dynamic.
- ② we can store heterogeneous type of data.
- ③ It is index'd type of collection.
- ④ It allows duplicate.
- ⑤ It follows order of insertion.
- ⑥ It allows null.
- ⑦ Since it is index'd type of collection, we can fetch the elements upon index randomly.
- ⑧ It is not auto sorted.
- ⑨ It increases its size by 50%

Date → 10/01/2023

```
★ import java.util.ArrayList;
public class Sample {
    public static void main (String[] args) {
```

```
        ArrayList l1 = new ArrayList();
```

```
        l1.add ("Hello");
```

```
        l1.add (123);
```

```
        l1.add (true);
```

```
        l1.add ('A');
```

```
        l1.add (12.5);
```

```
        l1.add (null);
```

```
        System.out.println ("Before" + l1);
```

```
        l1.add (2, "Yo");
```

```
        System.out.println ("After" + l1);
    }
}
```

O/P

Before [Hello, 123, true, A, 12.5, null]

After [Hello, 123, Yo, true, A, 12.5, null]

```
★ import java.util.ArrayList;
public class Sample {
    public static void main (String[] args) {
```

```
        ArrayList l1 = new ArrayList();
```

```
        l1.add ("Hello");
```

```
        l1.add (123);
```

```
        l1.add (true);
```

```
        l1.add ('A');
```

```
        l1.add (12.5);
```

```
        l1.add (null);
```

```
        System.out.println ("Before" + l1);
```

```
        ArrayList l2 = new ArrayList();
```

```
        l2.add ("Bye");
```

```
        l2.add (321);
```

```
        l2.add (false);
```

```
        l2.add ('B');
```

```
        l2.add (12.10);
```

```
        l2.add (null);
```

```
        System.out.println ("Before" + l2);
```

```
        l1.addAll (l2);
```

```
        System.out.println ("After" + l1);
    }
}
```

O/P

Before [Hello, 123, true, A, 12.5, null]

Before [Bye, 321, false, B, 12.10, null]

After [Hello, 123, true, A, 12.5, null, Bye, 321, false, B, 12.10, null]


```

★ import java.util.ArrayList;
public class Sample {
    public static void main(String args) {

```

```

        ArrayList l1 = new ArrayList();

```

```

        l1.add("Hello");

```

```

        l1.add(123);

```

```

        l1.add(true);

```

```

        l1.add('A');

```

```

        l1.add(12.5);

```

```

        l1.add(null);

```

```

        System.out.println("Before" + l1);

```

```

        ArrayList l2 = new ArrayList();

```

```

        l2.add("Bye");

```

```

        l2.add(321);

```

```

        l2.add(false);

```

```

        l2.add('B');

```

```

        l2.add(12.10);

```

```

        l2.add(null);

```

```

        System.out.println("Before" + l2);

```

```

        l1.addAll(3, l2);

```

```

        System.out.println("After" + l1);

```

```

    }
}

```

O/P →

Before [Hello, 123, true, A, 12.5, null]

Before [Bye, 321, false, B, 12.10, null]

After [Hello, 123, true, Bye, 321, false, B, 12.10, null, A, 12.5, null]

★ Package Collection topic;

```
import java.util.ArrayList;
```

```
public class Sample {
```

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

```
        ArrayList l1 = new ArrayList();
```

```
        l1.add("Hello");
```

```
        l1.add(123);
```

```
        l1.add(true);
```

```
        l1.add('A');
```

```
        l1.add(12.5);
```

```
        l1.add(null);
```

```
        System.out.println("Before" + l1);
```

```
        l1.remove(2); // Based on index.
```

```
        l1.remove(12.5); // Based on object.
```

```
        System.out.println("After" + l1);
```

3 }

O/P →

Before [Hello, 123, true, A, 12.5, null]

After [Hello, 123, A, null]

Date → 11/01/2023

★ QUEUE →

Queue is an interface whenever we want to follow genral queue then we should go for queue type of collection.

In Queue we have

① LinkedList

② Priority 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 cannot store heterogeneous type of data.
- ③ It is not index'd type of collection,
- ④ It allows duplicates.
- ⑤ It is auto sorted (Partially)
- ⑥ It does not allow null.
- ⑦ Since it is not index'd type of collection, we can't fetch the elements upon index randomly.
- ⑧ It does not follow order of insertion.

★ SET →

Set is an interface which helps to store the data not upon index and does not allow duplicates.

In set we have:-

- ① HashSet
- ② LinkedHashSet
- ③ TreeSet

★ features of SET →

- ① Size is dynamic.
- ② We can store heterogeneous type of data.
- ③ It is not index'd type of collection.
- ④ It does not allow's duplicate's.
- ⑤ It allow's null.
- ⑥ Since it is not index'd 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 heterogeneous type of data.
- ③ It is not index'd type of collection.
- ④ It doesn't allow's duplicate's.
- ⑤ It allow's null.
- ⑥ Since it is not index'd 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 packages.

★ features of LinkedHashSet →

- ① Size is dynamic.
- ② We can store heterogeneous type of data.
- ③ It is not index'd type of collection.
- ④ It doesn't allow's duplicate's.
- ⑤ It allow's null.
- ⑥ Since it is not index'd type of collection, we can't fetch the elements upon index randomly.
- ⑦ It increases its size by 50%.
- ⑧ It follow order of insertion.

★ features of TreeSet → It is a class which belongs to java.util packages.

- ① Size is dynamic.
- ② We can store only homogenous type of data.
- ③ It is not index'd type of collection.
- ④ It does not allow's duplicates.
- ⑤ It does not allow null.
- ⑥ Since it is not index'd type of collection, we can't fetch the elements upon index randomly.
- ⑦ It increases its size by 50%.
- ⑧ It is completely auto sorted.

class Cast Exception

Date → 14/11/23

```
★ import java.util.ArrayList;
public class Sample1 {
    public static void main (String[] args) {
        ArrayList l1 = new ArrayList();
        l1.add ("Hello");
        l1.add (123);
        l1.add (true);
        l1.add ('A');
        l1.add (12.5);
        l1.add (null);
        System.out.println ("Before "+l1);
        ArrayList l2 = new ArrayList();
        l2.add ("Hello");
        l2.add (1234);
        l2.add (true);
        l2.add ('B');
        l2.add (12.5);
        l2.add (null);
        System.out.println ("Before "+l2);
        l1.removeAll(l2);
        System.out.println ("After "+l1);
    }
}
```

O/p
Before [Hello, 123, true, A, 12.5, null]
Before [Hello, 1234, true, B, 12.5, null]
After [123, A]


```

* import java.util.ArrayList;
public class Sample1 {
    public static void main (String[] args) {
        ArrayList l1 = new ArrayList();
        l1.add ("Hello");
        l1.add (123);
        l1.add (true);
        l1.add ('A');
        l1.add (12.5);
        l1.add (null);
        System.out.println ("Before" + l1);
        ArrayList l2 = new ArrayList();
        l2.add ("Hello");
        l2.add (1234);
        l2.add (true);
        l2.add ('B');
        l2.add (12.5);
        l2.add (null);
        System.out.println ("Before" + l2);
        l1.retainAll (l2);
        System.out.println ("After" + l1);
    }
}

```

O/P →

Before [Hello, 123, true, A, 12.5, null]
 Before [Hello, 1234, true, B, 12.5, null]
 After [Hello, true, 12.5, null]

```

* import java.util.Vector;
public class Sample1 {
    public static void main (String[] args) {
        Vector<Integer> vector = new Vector();
        vector.add (10);
        vector.add (20);
        vector.add (30);
        vector.add (40);
        vector.add (50);
        vector.add (60);
        vector.add (70);
        vector.add (80);
        vector.add (90);
        vector.add (100);
        vector.add (110);
        int x = vector.capacity();
        System.out.println ("Capacity is" + x);
        for (int i=0; i<vector.size(); i++){
            System.out.println (vector.get (i));
        }
    }
}

```

O/P

Capacity is 20.

10
20
30
40
50
60
70
80
90
100
110


```

★ Package CollectionTopic;
import java.util.LinkedList;
public class Sample {
    public static void main(String[] args) {
        LinkedList l1 = new LinkedList();
        l1.add("1. Hello");
        l1.add("2. Hello");
        l1.add("3. Hello");
        l1.add("4. Hello");
        l1.add("5. Hello");
        System.out.println("*** Peeking ***");
        System.out.println(l1.peek());
        System.out.println(l1);
        System.out.println("*** Polling ***");
        l1.poll();
        System.out.println(l1);
        l1.poll();
        System.out.println(l1);
    }
}

```

O/P

[1. Hello, 2. Hello, 3. Hello, 4. Hello, 5. Hello]

*** Peeking ***

1. Hello

[1. Hello, 2. Hello, 3. Hello, 4. Hello, 5. Hello]

*** Polling ***

[2. Hello, 3. Hello, 4. Hello, 5. Hello]

[3. Hello, 4. Hello, 5. Hello]

```

★ import java.util.PriorityQueue;
public class Sample1 {
    public static void main(String[] args) {
        PriorityQueue p1 = new PriorityQueue();
        p1.add("Anil");
        p1.add("Birkhe");
        p1.add("Faloda");
        p1.add("Donkey");
        p1.add("ear");
        p1.add("Elephant");
        System.out.println(p1);
    }
}

```

O/P

[Anil, Birkhe, Elephant,
Donkey, ear, Faloda]

★ package collectionTopic;

import java.util.TreeSet;

public class Sample {

public static void main (String [] args) {

int [] arr = { 9, 1, 2, 4, 4, 5};

TreeSet < Integer> t1 = new TreeSet();

for (int i=0; i < arr.length; i++) {

t1.add (arr[i]);

System.out.println (t1);

}

O/P
→
[1, 2, 4, 5, 9]

Date: 16/01/2023

★ package collectionTopic;

import java.util.HashSet;

public class Sample {

public static void main (String [] args) {

HashSet set = new HashSet();

set.add ("Hi");

set.add (1234);

set.add ('A');

set.add (1234);

set.add ('A');

set.add (null);

set (set);

}

O/P
→
[null, Hi, A, 1234]

★ import java.util.LinkedHashSet;

public class Sample {

public static void main (String [] args) {

LinkedHashSet set = new LinkedHashSet();

set.add ("Hi");

set.add (1234);

set.add ('A');

set.add (1234);

set.add ('A');

set.add (null);

set (set);

}

O/P
→
[Hi, 1234, A, null]

* import java.util.TreeSet;

public class Sample {

public static void main (String args) {

TreeSet <Integer> set = new TreeSet();

set.add(24);

set.add(2);

set.add(12);

set.add(23);

set.add(21);

op(set);

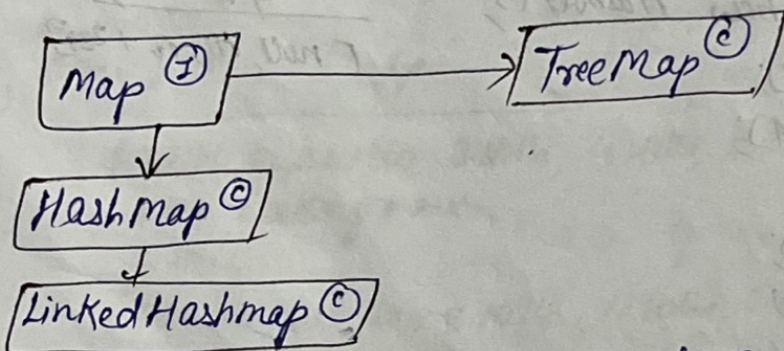
}

O/P

[2, 12, 21, 23, 24]

* MAP → Map is a Interface, which belongs to java.util package.

It is also a part of collection.



Map will allow the element Based on Key and value pair.

* HASHMAP → It is a class, which belongs to java.util package.

• It is a subclass of Map Interface.

Features of HashMap →

- ① It is a generic type.
 - ② Size is dynamic.
 - ③ The elements will be stored in the form of Key + value pair.
 - ④ Key cannot be duplicated.
 - ⑤ Value can be duplicated.
 - ⑥ It does not follow order of insertion.
- put is the method which is used to add the element inside the map.


```

* import java.util.HashMap;
public class Sample {
    public static void main (String[] args) {
        HashMap<String, Integer> h1 = new HashMap();
        h1.put ("Puma", 4000);
        h1.put ("Bata", 200);
        h1.put ("Nike", 4000);
        h1.put ("adidas", 6000);
        h1.put ("adidas", 8000);
        System.out.println (h1);
    }
}

```

o/p
 { Nike = 4000, Bata = 200, ~~Nike = 4000~~, adidas = 8000 }

* LINKED HASHMAP → It is a class, which belongs to java.util package.

It is a subclass of HashMap.

Features of LinkedHashMap →

- ① It is a generic type.
- ② Size is dynamic.
- ③ The elements will be stored in the form of key & value pair.
- ④ Key can not be duplicated.
- ⑤ Value can be duplicated.
- ⑥ It follows order of insertion.

```

* import java.util.LinkedHashMap;
public class Sample {
    public static void main (String[] args) {
        LinkedHashMap<String, Double> l1 = new LinkedHashMap<>();
        l1.put ("Bottle", 50.00);
        l1.put ("Duster", 45.90);
        l1.put ("Marker", 30.00);
        l1.put ("Board", 250.00);
        l1.put ("Board", 290.00);
        System.out.println (l1);
    }
}

```

o/p
 { Bottle = 50.0, Duster = 45.9, Marker = 30.0, Board = 290.0 }

* TREEMAP → It is a class which belongs to java.util package.

• It is a subclass of map interface.

features of TreeMap →

- ① It is a generic type.
- ② Size is dynamic.
- ③ The elements will be stored in the form of key & value pair.
- ④ Key cannot be duplicated.
- ⑤ value can be duplicated.
- ⑥ It is fully followed order of insertion based on key.
auto sorted.

```
* import java.util.TreeMap;  
public class Sample {  
    public static void main (String[] args) {
```

```
        TreeMap <String, Double> t1 = new TreeMap();
```

```
        t1.put ("Bottle", 50.00);
```

```
        t1.put ("Duster", 45.90);
```

```
        t1.put ("Marker", 30.00);
```

```
        t1.put ("Board", 250.00);
```

```
        t1.put ("Board", 250.00);
```

```
        System.out.println (t1);
```

```
    }  
}
```

o/p

{ Board = 250.0, Bottle = 50.0, Duster = 45.9, Marker = 30.0 }

★ Enhanced for loop → Here we go only forward direction.

Syntax:-
 for (ClassType ref-var : array/collection)
 {
 }
 }
 }
 }

★ Public class main {
 psvm(-) {
 String[] str = {"Hi", "Hello", "Bye"};
 for (String s1 : str)
 {
 sop(s1);
 }
 }
 }

★ import java.util.TreeSet;
 public class main {
 public static void main (String[] args) {
 TreeSet t1 = new TreeSet();
 t1.add(10);
 t1.add(20);
 t1.add(40);
 t1.add(30);
 t1.add(30);
 for (Object i1 : t1)
 {
 System.out.println(i1);
 }
 }
 }

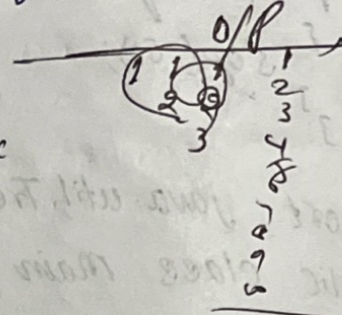
★★★★ To fetch the Object from the collection without doing downcast we should go for the Object type

Date: 22/01/2023

★ THREADS → Thread is a execution instance which owns it once CPU Time and memory.
Thread is a class which belongs to java.lang package.

★ Package ThreadTopic;

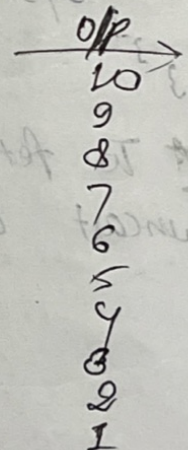
```
public class Sample {  
    public static void main (String[] args) throws InterruptedException {  
        for (int i=1; i<=10; i++)  
        {  
            System.out.println(i);  
            Thread.sleep(2000); // time 2 sec  
        }  
    }  
}
```



★ Demon Thread → The thread which is running in the backend is called as Demon thread.

- ★ We can implement a thread by two ways -
 - ① Through Runnable Interface.
 - ② By extending Thread class.

```
★ public class Sample implements Runnable {  
    public void run() {  
        for (int i=10; i>0; i--) {  
            System.out.println(i);  
            try {  
                Thread.sleep(2000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
    public static void main (String[] args) {  
        Sample s1 = new Sample();  
        Thread th1 = new Thread(s1);  
        th1.start();  
    }  
}
```



* MULTI-THREADING → The process of executing multiple Thread simultaneously at same time is called multithreading.

```
* class Demo extends Thread {  
    public void run() {  
        for (int i=1; i<=10; i++) {  
            System.out.println(i);  
            try {  
                Thread.sleep(3000);  
            }  
            catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}  
  
public class Sample implements Runnable {  
    public void run() {  
        for (int i=11; i>=5; i--) {  
            System.out.println(i);  
            try {  
                Thread.sleep(2000);  
            }  
            catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
  
    public static void main (String[] args) {  
        Sample s1 = new Sample();  
        Thread th1 = new Thread(s1);  
        th1.start();  
  
        Demo d1 = new Demo();  
        Thread th2 = new Thread(d1);  
        th2.start();  
    }  
}
```


* Runnable Interface →

Runnable interface is a functional Interface, which belongs to java.lang package.

* Difference b/w sleep wait () →

SLEEP()

WAIT()

- | | |
|--|---|
| ① It is a static method of thread class. | ① It is a non-static method of object class. |
| ② It will pause the execution for few milli-seconds. | ② It makes a thread to wait until it is a notification. |
| ③ It can be use in any case. | ③ It can be use only in a synchronized. |

* notify () →

It is a non-static method of Object class, which helps to notify the thread, which is about to access.

* notifyAll () → It is a non-static method of Object class, which help to thread to notify all which is about to access.

ArrayList

- ① It increases its size by 50%.
- ② It is not synchronized.
- ③ Since it is not synchronized the performance is fast.
- ④ Size is dynamic.
- ⑤ We can store heterogeneous type of data.
- ⑥ It is indexed type of collection.
- ⑦ It allows duplicates.
- ⑧ It follows order of insertion.
- ⑨ It allows null.
- ⑩ Since it is indexed type of collection we can fetch the elements upon index randomly.
- ⑪ It is not auto sorted.

Vector

- ① It increases its size by 100%.
- ② It is synchronized.
- ③ Since it is synchronized the performance is slow.
- ④ Size is dynamic.
- ⑤ We can store heterogeneous type of data.
- ⑥ It is indexed type of collection.
- ⑦ It allows duplicates.
- ⑧ It follows order of insertion.
- ⑨ It allows null.
- ⑩ Since it is indexed type of collection we can fetch the elements upon index randomly.
- ⑪ It is not auto sorted.

LinkedList

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

Priority Queue

- ① we cannot store heterogeneous type.
- ② It does not allow null.
- ③ It allows duplicates.
- ④ It does not follow order of insertion.
- ⑤ size is dynamic.
- ⑥ It is not indexed type of collection.
- ⑦ It is auto sorted.
- ⑧ Since it is not indexed type of collection we can't fetch the elements upon index randomly.

LinkedHashSet

- ① we can store heterogeneous type.
- ② It allows null.
- ③ It does not allow duplicates.
- ④ It follows order of insertion.
- ⑤ size is dynamic.
- ⑥ It is not indexed type of collection.
- ⑦ Since it is not indexed type of collection we can't fetch the elements upon index randomly.
- ⑧ It increases its size by 50%.

Tree Set

- ① we can store homogeneous type.
- ② It does not allow null.
- ③ It does not allow duplicates.
- ④ It is completely auto sorted.
- ⑤ size is dynamic.
- ⑥ It is not indexed type of collection.
- ⑦ Since it is not indexed type of collection we can't fetch the elements upon index randomly.
- ⑧ It increases its size by 50%.

HashSet

- ① we can store heterogeneous type.
- ② It allows null.
- ③ It does not allow duplicates.
- ④ It does not follow order of insertion.
- ⑤ size is dynamic.
- ⑥ It is not indexed type of collection.
- ⑦ Since it is not indexed type of collection we can't fetch elements upon index randomly.
- ⑧ It increases its size by 50%.

<u>Array List</u>	<u>Vector</u>	<u>Linked List</u>
• Here	Here	Here
• Order of marks	order given	order
• index type of data	index by	index by
• Not sym	symmetrical	
(Fast)	slow	
<u>Allowed</u>	Allowed	Allowed
<u>Null</u>	Null	Null
• Allowed duplicate	A-D	AD
• size sorted	increases	sort
• Not Auto sorted	Not Auto sorted	Not Auto sorted

<u>Binary Queue</u>
• Memory
Not order of insertion
Not index
Not Allowed Null
AD
sort
Auto sorted (partially)

<u>Hashset</u>	<u>Linked hashset</u>	<u>TreeSet</u>
Null	Null	• Memory
Not order of insertion	order of insertion	—
Not sym	Not index	Not sym
Allowed Null	Allowed Null	Not Allowed Null
Not AD	Not AD	Not AD
sort	sort	sort
—	—	Auto sorted (completely)

for array based approach we can use array list
for linked list based approach we can use linked list
for hashset based approach we can use hashset
for treebased approach we can use treebased approach