Saathi

Date ___/___/_____

Basic streams :-

1. System.in :- A logical stream (pipe) is connected with keyboard and object of scanner class.
2. System.out
3. System.err

~~Properties~~

Date 1, 8, 19

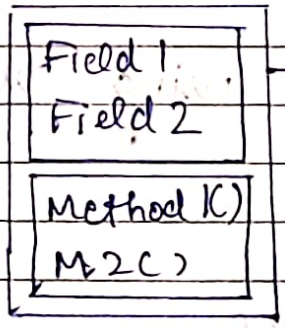
Saathi

OOP

Properties of OOP:-

1. Class and objects
2. Encapsulation
3. Abstraction
4. Inheritence
5. Polymorphism

Class:- It acts as a blueprint. It is collection of objects. It encapsulates fields and methods.



Object:- It is instance of class. It is a real world entity. Its state is defined by properties (fields) and behaviour is defined by action (method).

'new' is operator as well as keyword. It is used to create runtime memory for objects in 'heap' portion of RAM.

Date __/__/____

Saath

Class with main method :-

```
class Student {
    int id;
    String name;
```

instance Fields

```
    public static void main(String args[]){
        Student S1 = new Student();
/*
Note:- If fields are printed without initialise, default value will be printed as JAVA has 'garbage collector'. Default for int is 0, String is Null, float is 0.0
*/
    }
}
```

Above program using two classes :-

```
class Student {
    int id;
    String name;
}

class Main {
    public static void main(String args[]) {
        Student S1 = new Student();
        System.out.println(S1.id);
```

Page No.

'this' is a keyword in JAVA. It is a ref. variable which refers to current object.

Date __/__/____

Saathi

Object Initialization

There are 3 ways to Initialize :-

1. By using object ref variable. S1.id = 10
2. By using class method.
3. By using constructor.

2.

```
class Student {
    int id;
    String name;
    void init (int id, String name) {
        this.id = id;
        this.name = name;
    }
}
class Test {
    public static void main (String args[]) {
        Student S1 = new Student();
        S1.init (10, "abc");
    }
}
```

Instance Method

Terminologies

Mutator or setter :- Method To change the value

Accessor or getter :- Method to access the value

'new' is operator as well as keyword. It is used to create runtime memory for objects in 'heap' portion of RAM.

Saath

Class with main method :-

```
class Student {
    int id;
    String name;
}
```
} instance Fields

```
    public static void main(String args[]){
        Student S1 = new Student();
/*
Note: If fields are printed without initialization, default value will be printed as JAVA has 'garbage collector'. Default for int is 0, String is Null, float is 0.0
*/
    }
}
```

Above program using two classes :-

```
class Student {
    int id;
    String name;
}

class main {
    public static void main(String args[]) {
        Student S1 = new Student();
        System.out.println(S1.id);
```

'this' is a keyword in JAVA. It is a ref. variable which refers to current object.

Saathi

Date / /

Object Initialization

There are 3 ways to Initialize :-

1. By using object ref variable. S1.id = 10
2. By using class method.
3. By using constructor.

2.

```
class Student {
    int id;
    String name;
    void init (int id, String name) {
        this.id = id;
        this.name = name;
    }
}
class Test {
    public static void ma[illegible] args[]) {
        Student S1 = ne[illegible]();
        S1.init(
    }
}
```

Instance Method

Terminologies

Mutator or setter : value

Accessor or getter : [illegible] the value

Student S1 = new Student()

↓

Default constructor is called.

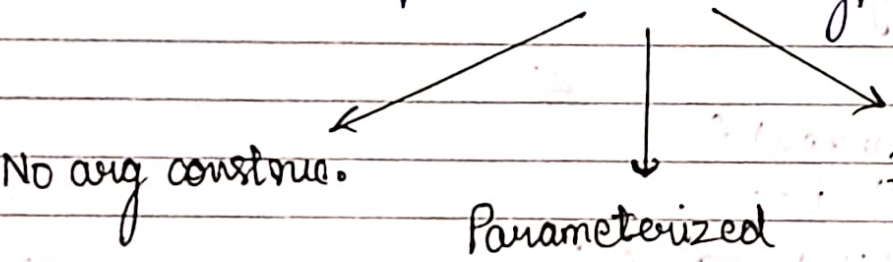
Saathi

Date __/__/____

3. Constructor

It is used to Initialise the object and it is called when object is created using 'new'.

Constructor is block of code but name of constructor is same as class name and there is no explicit return type.



Scanner scan = new Scanner(System.in)

Parametrized constructor is called

Q WAP to initialise object through constructor.

```
class Student {
    Int id;
    String name;
```

Page No.

More than one constructor possible : Constructor Overloading

Saathi

Date __/__/____

```
Student ( int id, string name) {
    this. id = id;
    this. name = name;
  }
}

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

        Student S1= new Student (20, "abc");
    }

}
```

Note:-

1. Instance variables/methods are stored in object memory i.e. heap area of RAM.
2. Static variables/methods are stored in class memory.

```
class Student{
    static String college= "MNNIT";
    int id;
    String name;
}

// in main()
    Student. college; // MNNIT
    S1. college;      // MNNIT
    S1. college = "IIIT"; // Changes will be reflected everywhere
                          // unlike Python.
```

Page No.

Date 8/8/19

Saathi

Uses of 'this' keyword

1. It is used to refer current object's instance variable.

2. 'this' is used to invoke current class constructor.

Example:-

```
class student {
    int id;
    string name;
    double fee;

    Student () {
        this (10, "abc");
        this (2080.20);
    }
    Student (int id ; string name){
        this.id = id;
        this.name = name;
    }
    Student (int id , String name , float fee){
        this( id, name);
        this.fee = fee;
    }
```

3. 'this' is also used to reuse the constructor.

4. 'this' can also be used to invoke current class method.

Page No.

nextLine() → String
next() → single word string

Date ___/___/___

Example:-

```
class Student {
    int id;
    String name;
    float fee;
    void display_name();
    { SOP(name); // Sys.out.pr(name); }

    void Input (String name)
    {
        this.name = name;
        this.display_name();
        // display_name();
    }
}
```

5. 'this' can be passed as an argument to constructor / method calling.

Example:-

```
class Test {
    int a, b;
    Test()
    { // Initialise using scanner.
    }
    void display(Test obj)
    { Sys.out.prin("a=" + a + "b=" + b); }
    void get()
    {
        display(this);
    }
}
```

Date

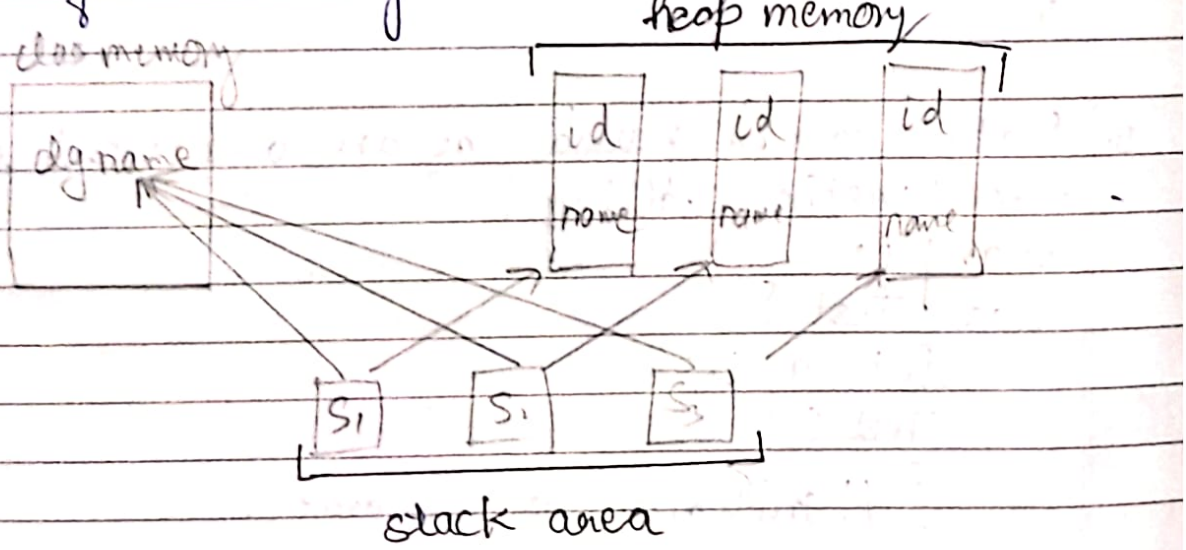
static keyword

It can be applied to Instance variable/ method.

instance variable — static → class variable
instance method — static → class method

```
class Student {
    int id;
    String name;
    static String clgname = "MNNIT";
}
```

* After creating S1, S2, S3



Rules for 'static' keyword :-

1. Static method belongs to class rather than object.

2. Static method can be invoked without creating object (i.e. saves memory).

Saathi

Date ___/___/_____

Static method can access static data-member

Static method CANNOT use non-static data-member directly.

'this' can't be used in static context.

Java Array

Java array is an object which contains similar type of data items.

Declaration datatype arrayName [size]; // Memory NOT allocated yet.

Eg:- int arr[] = {3, 4, 5}

datatype arrName[] = new datatype[size];
↳ Memory is allocated

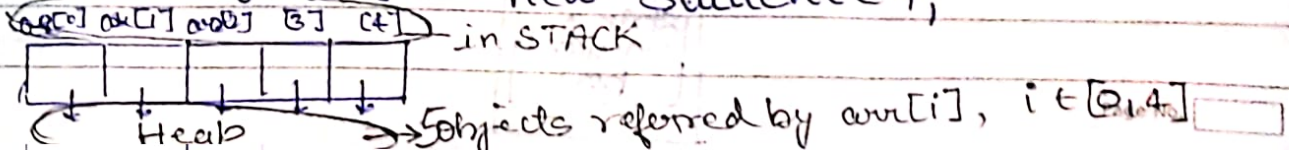
For each loop:-

```
for (datatype var : arrName) {
    //body
}
```
} Syntax

Eg:-
```
for (int i : arr) {
    SOP(i);
}
```

// Assume student class.

```
Student arr[] = new Student[5] // Array of Student class
for(int i=0; i<5; i++)
    arr[i] = new Student();
```



Saathi

Date

// Assume private field stipend in student class.

```
int getMaxStipend(student S[]){


}
// Calling
   getMaxstipend(S, ovr);
```

Returning array from method:-

```
datatype [] methodName (arguments){
            //body
}
```

22/8/19

String

String is an object of class String that represents sequence of character.

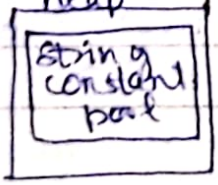
Two ways to create String

1. String literal
2. new keyword

```
String S = "hello";
```

S → reference variable in stack. "hello" → object memory created in heap area and it is called 'String constant pool'.

```
String S1 = new String("hello");
```

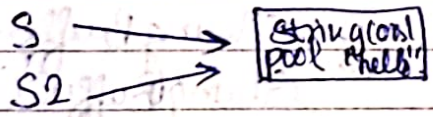


Page No.

Date ___/___/_____

Saathi

* In case 1, if we write
String S2 = "hello";
New object "hello" will NOT be created as JVM searches 'String constant pool' if "hello" is found S2 will point to previously made "hello".



* In case 2, new object will be created every-time.

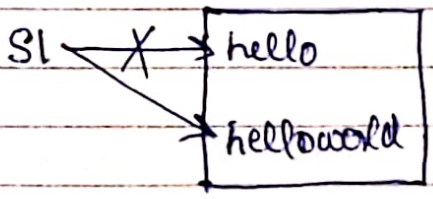
Some String methods:-
String S1 = new String ("hello");

1. char charAt(int index)

S1.charAt(2); //Returns 'l'

2. int length() int len = S1.length()

3. String concat(String str)
S1 = S1.concat("world");



Note:- String class object is immutable in nature

Page No.

Synchronized : This method is now 'thread safe'. Second process can't start until ongoing ends
Significance In Multithreading (Chat Application)

String Buffer Class

Object of this class is mutable in nature.

Object can be created using following constructo

(i) StringBuffer() // Empty String
(ii) StringBuffer(String str)

Example:-

```
StringBuffer S1 = new StringBuffer();
StringBuffer S2 = new StringBuffer("hello");
```

Some methods:-

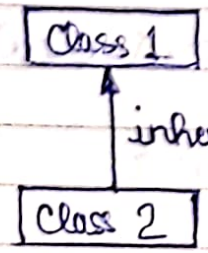
1. public synchronized String append(String str)

```
S2.append("World");
```

Date 22, 8, 19

Saathi

Inheritance



Parent / Super

Child / Sub

```
class class1 {
        d1;
        d2;
        m1();
        m2();
}

class class2 extends class1 {
        d3;
        m3();
}
```

- This feature is called 'CODE REUSABILITY'.

class2 obj = new class2(); Line 1

'obj' can access all 3 datamembers and 3 methods

Line 1 will firstly call constructor of class1 as its object is created and then constructor of class2 is called.

Page No.

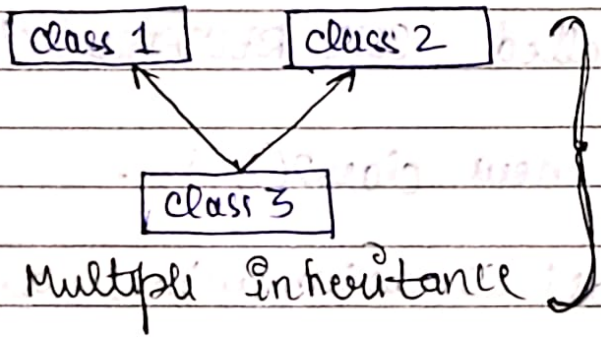
Saath

Write Output :-

```
class Person {
    person ( ) {
        SOP(" Object of Person is created");
}
class Student extends Person {
    student( ){
        SOP(" student object is created");
}
class Test {
    public static void main(String args[]){
        student S= new Student();
    }
}
```

Output :-

Object of Person is created
student Object is created.



Write output.

```
class one{
    Int i=10;
    one ( ){ }
    void show( )
    { SOP ("Super class=" +i); }
}
```

super:- refers to parent class.

Saathi

Date __/__/____

```
class Two extends One {
        int i = 20;
        Two() { }
        void show()
        {
            cop ("Sub class = " + i);
            super.show();
        }

}
class Test {
    public static void main( String args[]){
                                Two.
        Two t = new ();
         t.show(); // By default calls child show()
}
```

Output

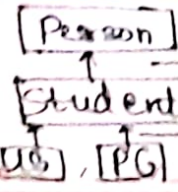
Sub class = 20

Super class = 10

* Super calling constructor.

```
class Person {
    int age;
    String name;
    person() { }
    person (int age, String name) {
        this.age = age;
        this.name = name;
    }
}
class Teacher extends Person {
    double salary;
    Teacher (int age, String name, double salary)
```

Page No.



Student IS A part of Person.

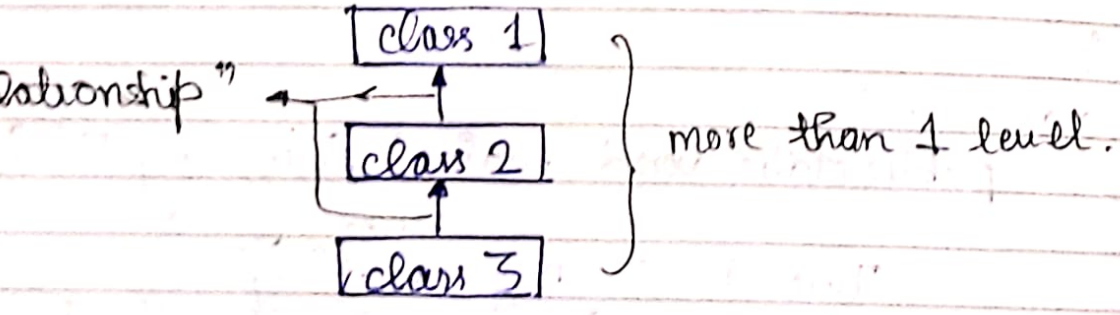
PG, UG IS A part of Student.

Saathi

```
{
    this.salary = salary;
    super(age, name);
  // Constructor of Person is called.
}
```

super must be 1st statement.

Multilevel inheritance



29/8/19

Code:- Tell output

```
class X
{
    public X(int i)
    {  SOP("1"); }
}
class Y extends X {
    public Y()
        System.OP("2");
}
class Test {
    public static void main(){
        Y obj = new Y();
    }
}
```

Date ___/___/_____

Saathi

Output

(i) | 1 |
    | 2 |

(ii) | 2 |

(iii) Error

Since default construct of X can't be called as parameterized constructor already exists and no parameter is passed to call parameterized constructor.

Output ??

```
class X {
    X() {
        SOP ("A");
    }
}
class Y extends X {
    Y() {
        SOP("B");
    }
}
class Z extends Y {
    Z(){
        SOP(" C");
    }
}
class Test {
    main(){
        Z obj = new Z();
        Y obj1 = new Y();
    }
}
```

| A |
| B |
| C |

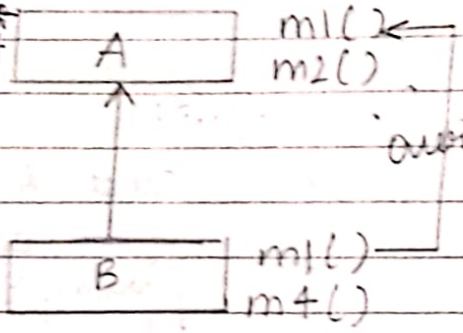
| A |
| B |

Signature – Method name + parameters
void m(int a){
signature

Saathi

Date ___ / ___ / ____

Method overriding



Used to redefine a method(s) already present in parent class according to its needs.

Polymorphism

poly + morphs
many forms

In JAVA it can be done in two ways

Method Overloading

Condition:
- More than 1 methods having same name (in a class).
- All the methods must have different signature.

This is Runtime Polymorphism OR Dynamic Polymorphism/Binding.

Method Overriding

It happens in Inheritance

Page No.

Note:- Field is NOT overrided ONLY methods are.

Date __/__/____

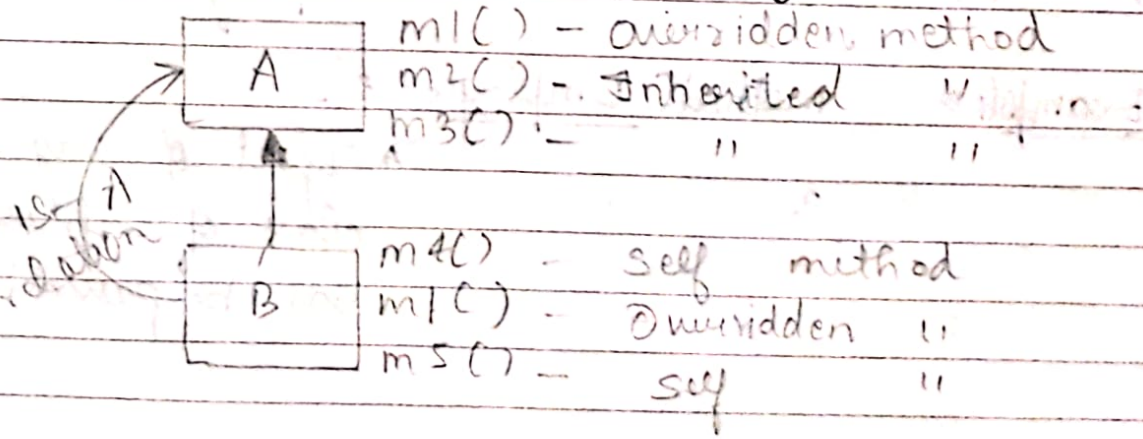
Saathi

```
class X{
    void m(int a)
     { - - }
    void m(int a, int b)
     { - - . }
}
class Test{
    main(){
        X obj = new X();
obj created at Runtime
        X.m(6,5);
```

thus above is Runtime polymorphism.

```
class X{
    static void m(int a){ }
    static void m(int a, int b){ }
}
class Test{
    main(){
        X.m(5); // At compile time
} //Thus static/compile time polymorphism.
}
```

Method Overriding



Date ___/___/_____

Saathi

Case 1

A obj = new A();

obj. | m1() | → of parent (A).
| m2() |
| m3() |

Case 2 → This is instance of A as well as B.

B obj = new B();

obj. | m1() | → of child (B)
| m2() |
| m3() |
| m4() |
| m5() |

Polymorphism

* Child class object can also be refered by object reference of parent class.

Employee

Manager

· Generalization

Employee e = new Manager();

Manager m = new Employee(); // Wrong

Example:- method(Employee e){

}

↑ Object of any child class of Employee can be passed.

Page No.

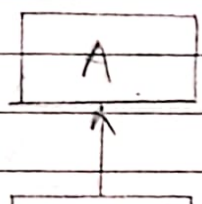
instance of → operator ( ) – cast operator

Date __/__/__

if(e instance of Manager) } True(Both)
if(e instance of Employee) }

~~Parent class ob~~

Employee e = new Manager(); } Parent class reference ~~object~~ refers to child object
" e1 = " Clerk();
" e2 = " Engineer();



A obj = new B();

Runtime/Dynamic Binding { obj. m1(); → Latest definition called i.e. of B, ~~can be both of A or B~~
obj. m3(); X

Upcasting / widening

RBI ob1 = new SBI();

RBI ob2 = new PNB();

Type conversion

Implicit (Widening) (Upcasting)

Explicit (Narrowing) (Downcasting)

Specialization

Manager m = (Manager) new Employee();

↳ Compiler passes this
Run time error arises → ClassCastException arises.

So is this possible? Search Yourself.

Page No.

Saath

Date

Access specifiers (Inheritance):-

Restriction ←

Visibility (within)	private	default	protected	public
~~Visibility~~ class	✓	✓	✓	✓
package	X	✓	✓	✓
outside package	X	X	✓ (if inherited)	✓

in overiding

Overridden	Protected	Default
Super if → Public m() Sub then → Public	Public Protected	Public Protected default

'Final' keyword

1. 'Final' keyword prevents inheritance.

```
final class A { }
```
This class's subclass can't be made.

2. To make variable constant.

```
double
Final PI = 3.14;
```

Page No.

Date 5 / 9 / 19

Saathi

```
class Derived {
    protected final void getDDetails() {
            SOP ("derived class");
    }
}
public class Test extends Derived {
    protected final void getDetails() {
            SOP ("Tst class");
        main(String a[]) {
            Derived obj = new Derived();
            obj.getDetails();
        }
    }
}
```

Output
Error

Note:-
Final method can't be overridden.
Final method can be inherited.

Benefits of generalization:-

1. Polymorphic array:-

```
Employee e[] = new Employee[5];
e[0] = new Manager();
e[1] = new Clerk();
e[2] = new Enginer();
```

e[0].getSalary → overridden method of Manager.

Page No.

Saath

Date ___/___/_____

2. Polymorphic method parameter :-

```
void calPension (Employee e);
```

Down casting (correct way):-

```
Employee e = new Manager();
Manager m  = (Manager) e;
```

m' (state)
m
m3

```
Employee e = new Manager()
```

m' (state)
m2
m4

e. { m1 → of parent

Note:- State methods ~~are not~~ does not participate in polymorphism as it is not part of object.

Abstraction

Implementation details are hidden.

abstract class (0 - 100% abstraction)
→ No concrete method
↓
No abstract method

interface (100% abstraction)

Date __/__/____

Saathi

100% abstract class ≠ interface

1. abstract class :- 'abstract' keyword is used.

concrete methods :- normal methods - defined
abstract methods :- not defined.

```
abstract class Test {

    abstract void show();  - no definition
    void print() {
        SOP("hello");
    }
}
```

* Abstract class can have 0 or more abstract method.
* If a class is having atleast 1 abstract method then it must be abstract.

To write definition of abstract methods; we need to override it in child class.

```
class Test1 extends Test {
    void show(){
        SOP("hi");
    }
}
```

* abstract class can't be instanciated. using new keyword.

Test obj = new Test(); ⇒ Error

Page No.

Date __/__/____

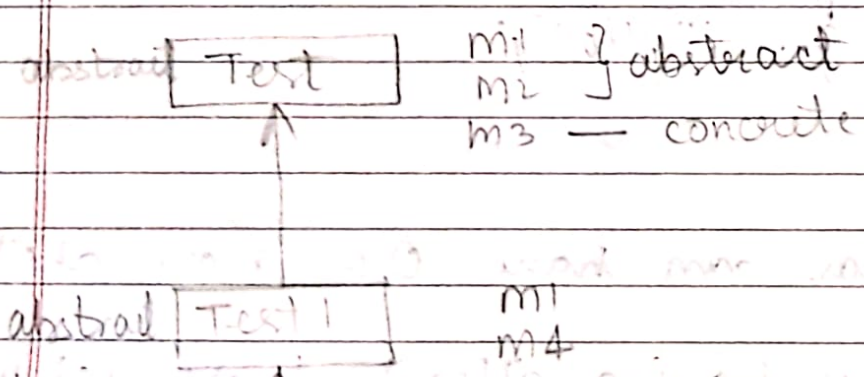
Test1 obj1 = new Test1();

obj1. { show() - overridden method
print() - inherited "

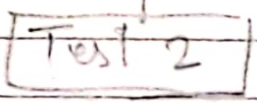
* Reference of abstract class can be created.

Test obj2 = new Test1(); Correct

obj2. { show()
print()



In Test1, we have an inherited abstract method m2() which is NOT overridden, thus an error will arise unless we make Test1 'abstract'.



Test2 obj = new Test2();

obj. { m1 → inherited from Test1 (overridden by Test1)
m2 → overridden
m3 → inherited from Test
m4 → inherited from Test1

Test1 obj1 = new Test2(); Correct

Page No

Date __/__/____

Saathi

* abstract class can have :-
(i) abstract method
(ii) concrete method
(iii) static concrete
(iv) final concrete — can't be overridden

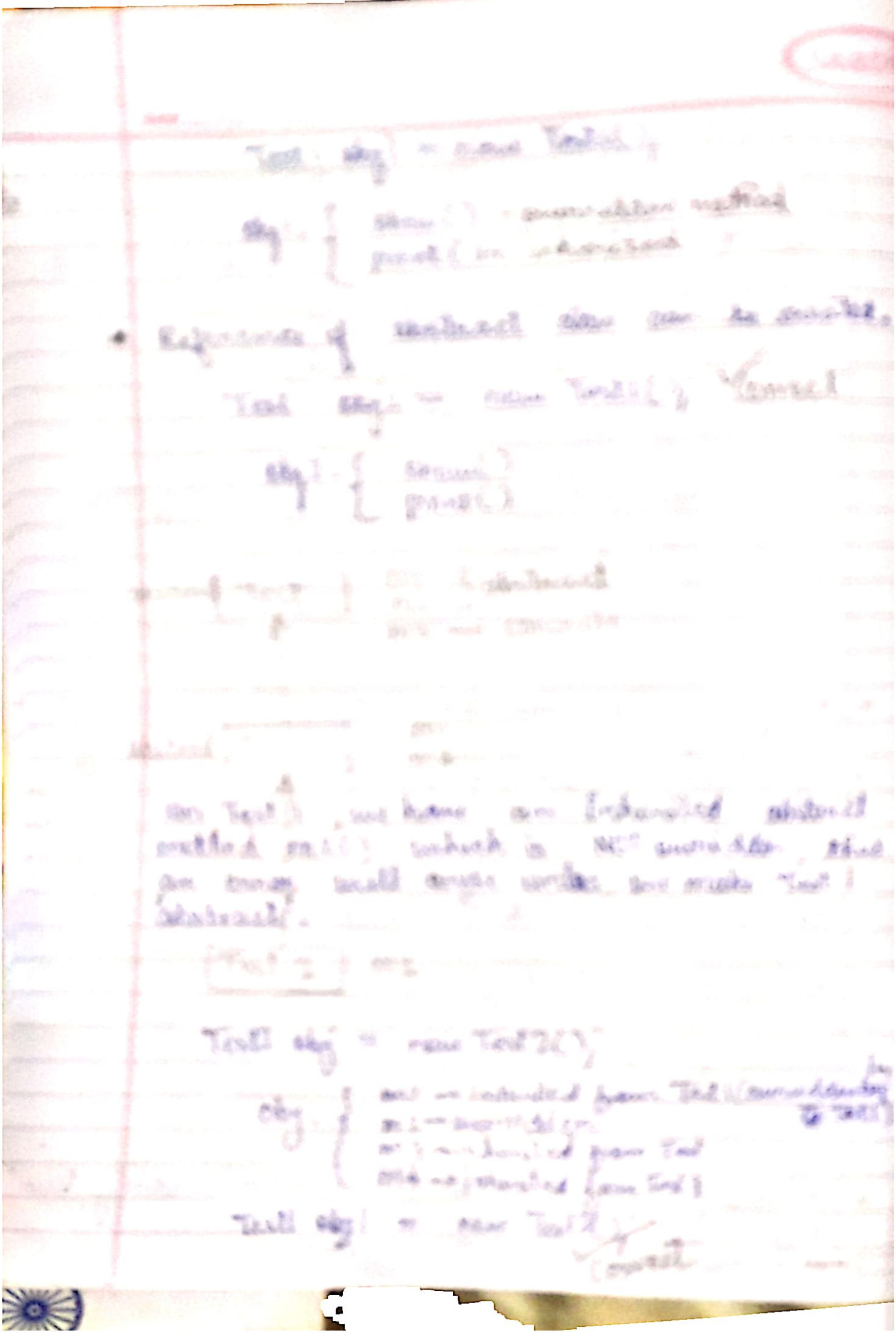
Note:- Don't use 'final' with 'abstract method' otherwise it won't be overridden.

Why to use abstract class/method at all?
1. If we leave method definition empty; in normal class memory is wasted as object of class can be created.
2. Confusion is sorted.

Q Output :-

```
abstract class Super {
    abstract int calc(int i);
    public static void show() {
        SOP("hello");
    }
}
class Sub extends Super {
    public int calc(int i) {
        return i;
    }
}
class Test {
    main() {
        Sub s = new Sub();
        SOP(S.calc(5));
        Super S1 = new Sub();
        S1.show() // Correct
    }
```

Page No.



abstract
abstract method
concrete method
static concrete
final concrete - cannot be overridden

note: we can use 'final' with 'abstract method' either
it will be overridden

why to use abstract class/ method ?

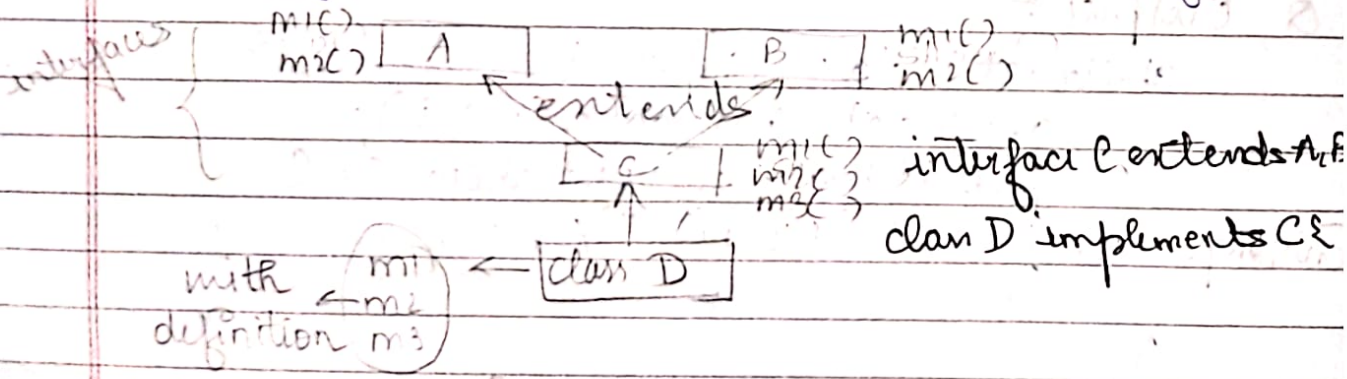
Scanned by CamScanner

Deadly diamond problem

Date 9 2 19

Java interface:- It provides 100% abstract

1. 'interface' is a keyword to declare interface.
2. All the methods in interface are by default public and abstract. No need to write these keywords.
3. We can't instantiate the interface.
4. But the object reference variable may be of type interface.
5. By default the data-member of interface are public static final.
6. An interface can't extend a class. but can extend another interface.
7. An interface can extend multiple interface.

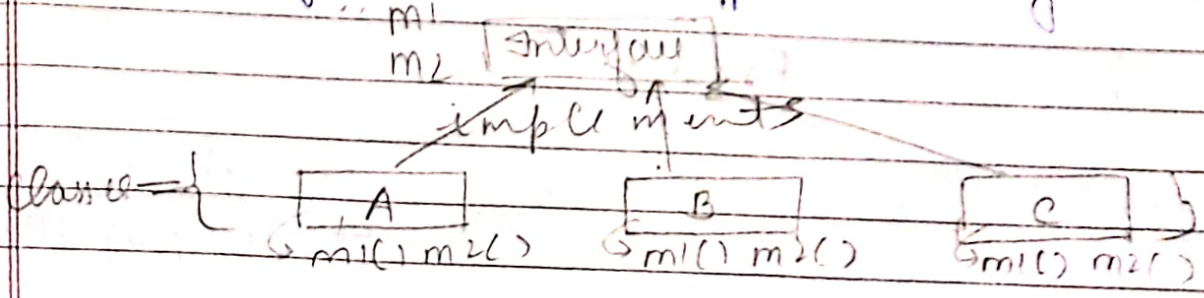



```
interface A {
    void show();  //public

}
class B implements A {

    public void show(){
        SOP("hello");
    }

}
class Test {
    main(){
        A obj = new B();
        B obj = new B();
```

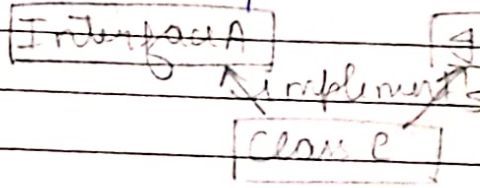
Date __/__/____

Saathi

8. Any no. of class can implement single interface.



9. A class can implement multiple interfaces.



class C implements A, B

Thus, indirectly, 'multiple inheritance' is possible

10. class A extends B implements C, D

Example:-

```
interface A{
    int l=5; = public static final int l=5;
    public void print();
}
interface B {
    public void show();
}
interface C extends A, B {

}
class Test implements C {

    public void print(){
        SOP("Hello");
    }
    public void show(){
        SOP("World!");
    }
}
```

Page No.