Paper Source
Subject.....
Date :.....Time:.....

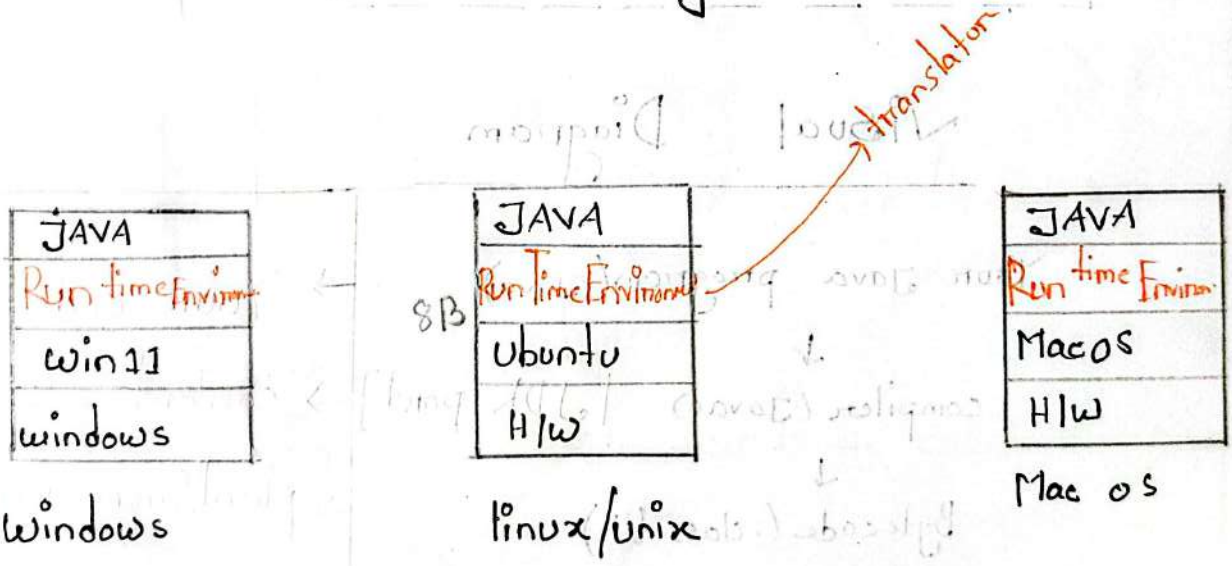
History at JaVa

Creator

James Grosling at Sun MicroSystem 1990s

WoRA,

Write Once Run Anywhere.



Windows linux/unix Mac os

JRE = JAva Run time Environment : that provides jvm + necessary libraries.

JRE = JVM + Libraries + other tools

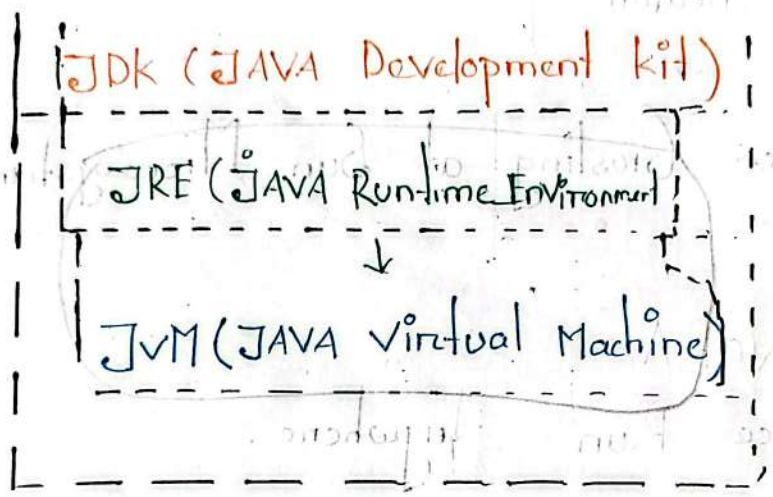
JVM = JAVA Virtual Machine → inside JRE acts like Translator

JVM = Translate Bytecode (.class file) into Machine code.

Paper Source
Subject.....
Date :.....Time:.....

Relationships between (JDK, JRE, JVM)

Diagram



Visual Diagram

Your Java program(.java) → Your code

↓

compiler (Javac) [JDK part] → convert code to Bytec

↓

Bytecode (.class file) → platform - independent co

↓

JVM (Translator) [Inside JRE] → Translate bytecode Machine code.

↓

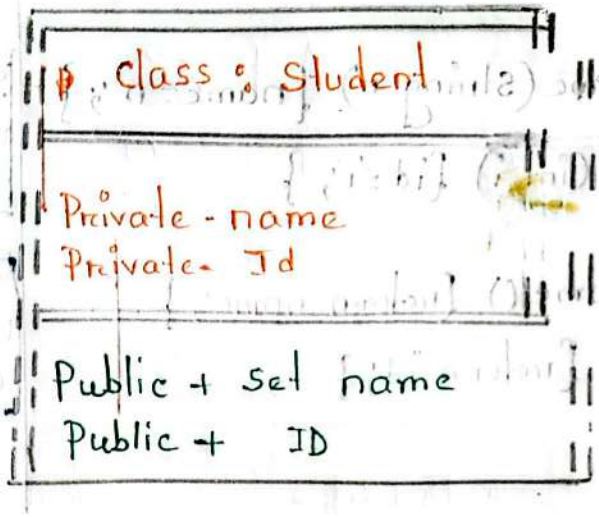
Machine code → OS → Hardware → (PROGRAM)

JAVA is A object oriented

Paper Source
Subject.....
Date :.....Time:.....

Principles OSS -> OPP

(i) Encapsulation :-> Bundles data and methods into one unit object.



Data + Methods in one class + controlled access = Encapsulation.

(i) Data Security : private variables prevent unauthorized access and accident change

(ii) controlled Access : Getter, Setter methods allow to control reading and updating data

changes in variables or methods used only the class no external code. object can be easily.

Paper Source
Subject.....
Date :.....Time:.....

```
impont java.util.Scanner;

class Student {
    Private String name;
    Private int id;

    Public void setName(String n) {name=n;}
    Public void Set_Id(int i) {id=i;}

    public String getName() {return name;}
    public int getid() {return id;}
}

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

        Scanner sc= new Scanner(system.in);
        Student s= new Student();
        System.out.print("Enter Student Name:");
        s.setName(sc.nextLine());
        System.out.print("Enter Student ID:");
        s.setId(sc.nextInt());
        System.out.println("\n Student Details:");
        System.out.println("Name:"+s.getName()+
                           "ID:"+s.getId());
    }
}
```

Input from keyboard

create class

private Variable
This is Encapsulation

Setter methods used to change data

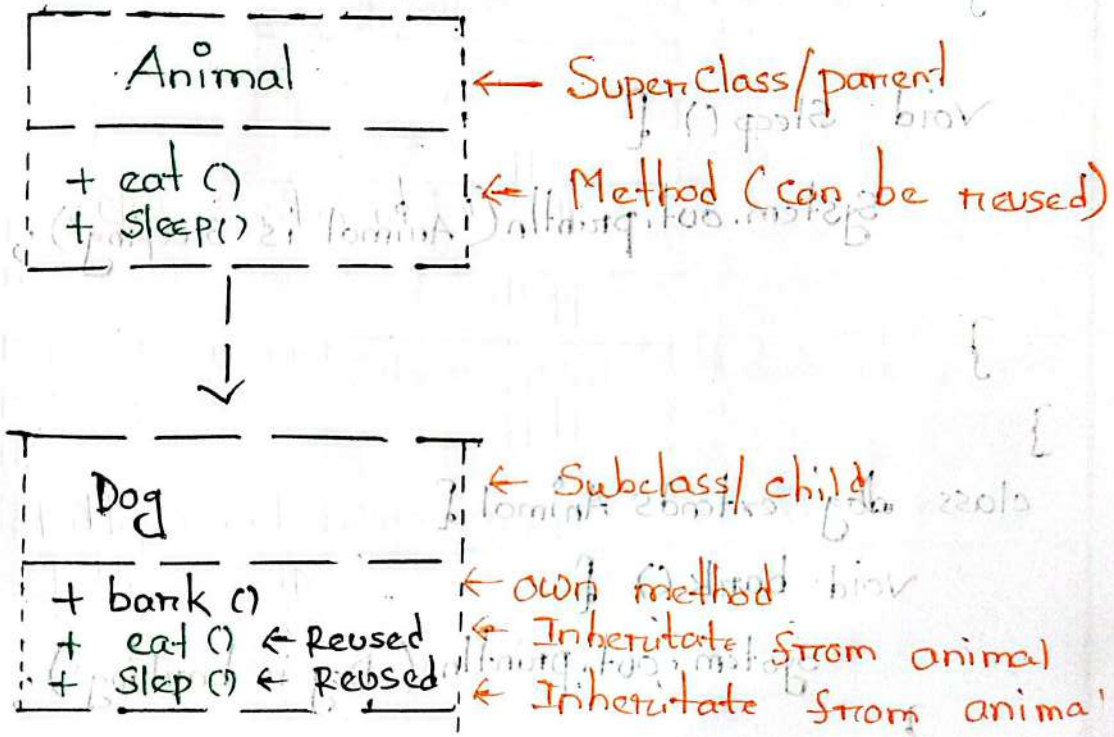
Getter methods used to read data

Paper Source
Subject.....
Date :.....Time:.....

(i) Inheritance — Subclass inherits properties & methods from Superclass → Reuse & extend code easily.

AS. like, DNA

Diagram:



(i) code Reuse : (i) Method, variable of the parent class → reused in, Subclass.
(ii) No need write Same code again.

(ii) Extensibility : New Sunctionality can be easily added by creating subclass.

(iii) Subclass can modisy or customize methods of the parent class.

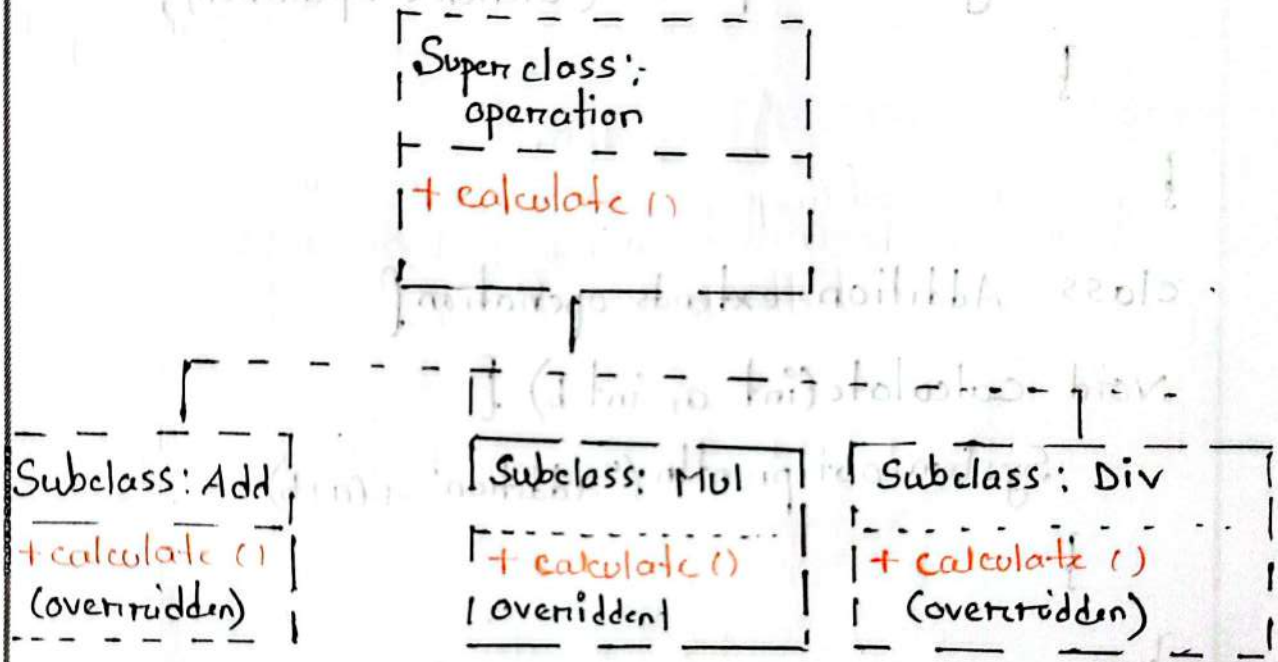
creates a parent - child structure → makes code organized and logical.

change pc → asfect all Subclass.

```
class Animal {
    void eat () {
        System.out.println ("Animal is eating");
    }
    Void Sleep () {
        System.out.println("Animal is Sleeping");
    }
}
class dog extends Animal {
    void bark () {
        System.out.println ("Dog is barking");
    }
}
public class main {
    public Static void main ( Strings [] args) {
        Dog d = new dog ();
        d.eat();        // in herited Method
        d.sleep();      // inherited Method
        d.bark();       // Dogs own Method
    }
}
```

Paper Source
Subject.....
Date :.....Time:.....

3. polymorphism :- One methode name, many behaviors



```
class operation {
    void calculate (int a, int b)
        System.out.println ("Generic operation);
    }
}

class Addition extends operation {
 void calculate (int a, int b) {
     System.out.println ("Addition: "+(a+b)
    }
}
class Multiplication extends operation {
    void calculate (int a, int b) {
      system.out.println ("Multiplication: "+(a*b);
   }
}  public class main {
     public static void main (String [] args) {
    operation op1 = new Addition ();
    operation op2 = new multiplication ();
    op1.calculate (10,5)
  { op2.calculate (10,5)
```

Paper Source
Subject
Date : Time:

polymorphism:-

Abstraction :->
- Hide implemention
- Expose essential fe

```
abstract class AtmSystem {
abstract void withdrawMoney (int amount);
}

class MyBankATM extends ATMsystem {
  Void withdrawMoney (int amount) {
      System.out.println ("Verifying pin...");
      System.out.println (" connecting to the bank Server");
      System.out.println ("Withdrawing Tk" + amount);
  }
}
    public class Main {
        public Static void main (String [] args) {
            ATMSystem atm = new My-Bank_ATM ();
      atm.withdrawmoney (5000);
    }
}
```

subclass must implement abstract methods

Paper Source
Subject.....
Date :.....Time:.....

1°÷ Cannot °instantiate°÷ Cannot create object of abstrac class.

```
ATMSystem atm = new ATMsystem();  ✗
Abstract class cannot have its own object.
```

But you can create reference of an abstract class ar assign concrete Subclass using Extend

Extend °÷ parent

child = new box

child extend parent =

child extend parent = child take everything from parent.

Code Visualization

parent Box (ATMSystem) [abstract]

withdrawMoney () (abstract)

extends

MyATMBANK

withdrawMoney | implement

depositMoney | extra feauture

Reference

MyBankATM obj

withdrawMoney.

Pass reference object

Declar ATMsystem = new My

REDMI NOTE 13 SHAKIB HOSSAN

CamScanner

Paper Source
Subject:
Date : Time:

2. Abstract class can have abstract methods.

```
abstract class ATMSystem {
    abstract void withdrawMoney(int amount);
}
```

No body only name and declaration

3. Subclass must implement abstract methods.

4. Abstract class have abstract Method
also have normal Method.

Mathod implement

Methode	implement	child ?
Abstract Methode	Subclass	Yes ✓
Normal Methode	Abstract class	NO ✗

5. construction can exist,

```
abstract class ATMSystem {
    ATMSystem() { ---- construction
    ....
    ATMSystem atm = new ATMSystem MyBankATM(), pa
```

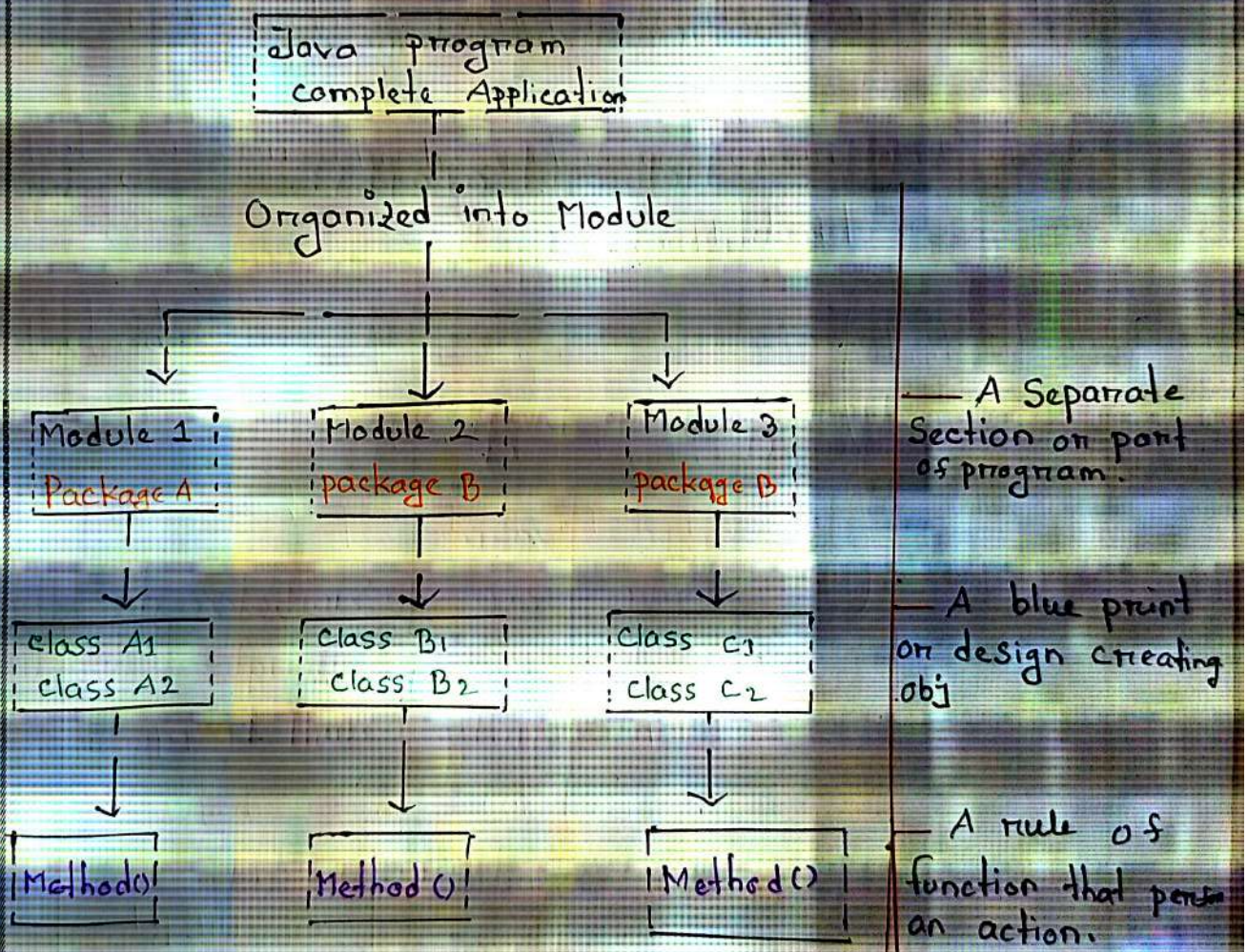
Ressurance abstract

Java Modularity

Paper Source
Subject.....
Date :.....Time:.....

Modularity ⟶ dividing a program into Small, independent parts.

Java Modularity concept Diagram



Java Data Types

Paper Source
Subject.....
Date :.....Time:.....

Primitive data type.

boolean	~ 1 bit	true/false. logic & condition
byte	1	Smallest integer, tiny numbers
char	2	character, unicode, arithmetic possible.
short	2	Small integer, Moderate numbers
int	4	Standard integer, most used
float	4	Medium precision, Smaller Memory fractional value
long	8	large integer, big num
double	8	high precision, math function, big numbers.

BcS. Fild

Paper Source
Subject.....
Date :.....Time:.....

Dynamic Initialization.

Run time declaration

```
class DynInt {
  public static void main (String args []){
    double a = 3.0, b = 4.0
    double c = Math.sqrt (a*a+b*b)  ——— Dynamic
    System.out.println ("Hypotenuse is"+c);
  }
}
```

Memory Flow Diagram

Variable	Type	Value	
a	doube	3.0	→ initialize directly
b	double	4.0	→ initialize directly
c	double	? not yet	→ will be computed dynamically

Program Run = Math.sqrt (a*a+b*b)

Step 1: a*a = 3.0 * 3.0 = 9.0
2 b*b = 4.0 * 4.0 = 16.0
3 a*a+b*b = 9.0 + 16.0 = 25.0
Math.sqrt = 5.0

c = 5.0

Block and Scope

Paper Source
Subject.....
Date :.....Time:.....

Block → code enclosed in {... } defining own Scope.

Scope

class Scope [instance/static Variables] → call just

- Methode Scope [local variables inside methods]
  - Block Scope [inside {...}]
  - Loop Scope [for, while, do]

Methode Scope everywhere.

Short Example

```
class Example {
    public static void main(String args[]) {
        int x = 10;
        {
            int y = 20;
            System.out.println(x);
            System.out.println(y);
        }
        System.out.println(x);
        (y) ✗
    }
}
```

→ Methods

class

not print

Nested Scope

Paper Source
Subject.....
Date :.....Time:.....

* Block inside another block.

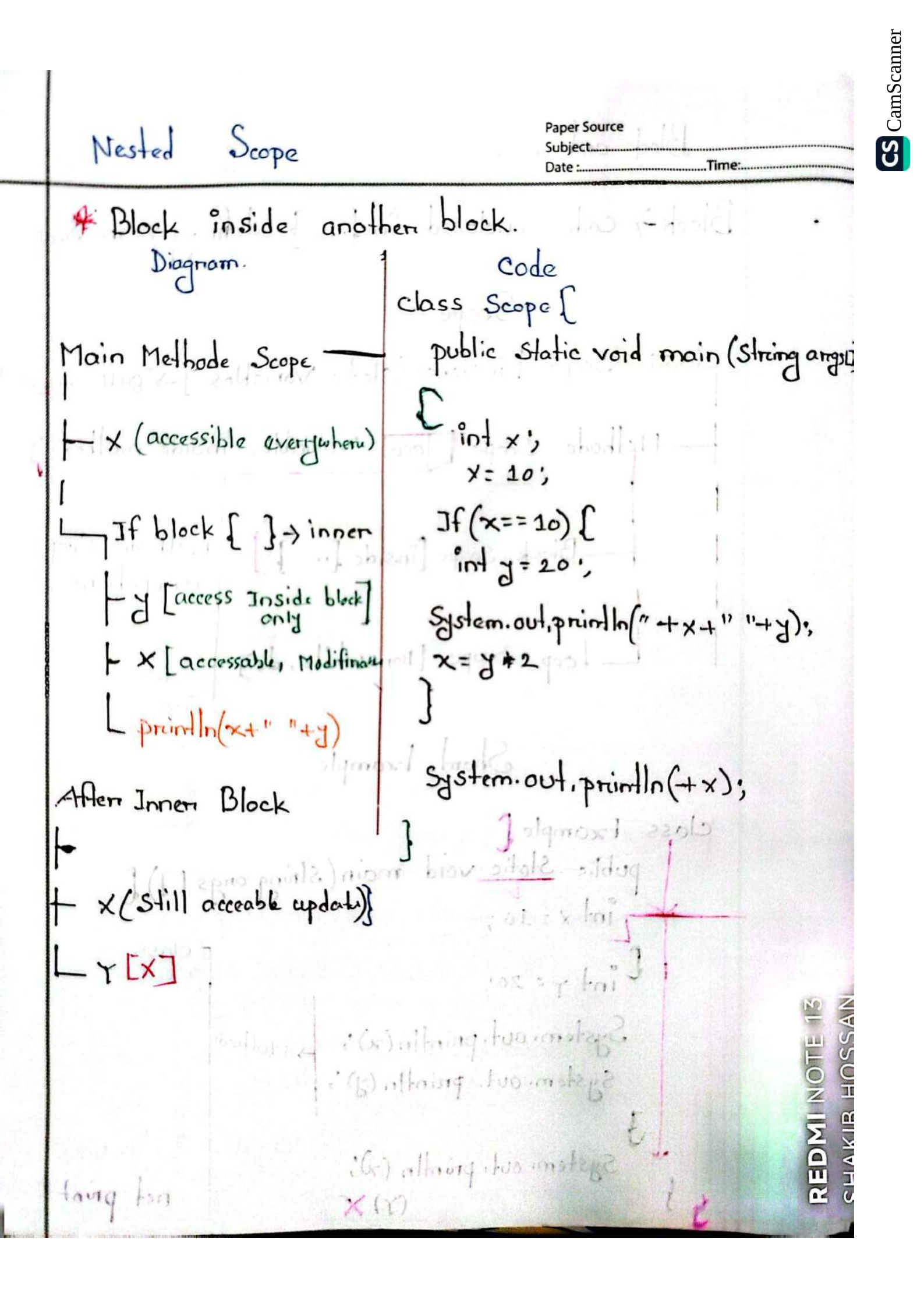
Diagram.

Main Methode Scope
|
├ x (accessible everywhere)
|
└ If block { } → inner
  ├ y [access Inside block only]
  ├ x [accessable, Modifinable]
  └ println(x+" "+y)

After Inner Block
|
├
├ x (Still acceable updated)
└ Y [x]

Code

```
class Scope {
  public static void main (String args[])
  {
    int x;
    x = 10;
    If (x==10) {
      int y = 20;
      System.out.println(" +x+ " "+y);
      x = y * 2
    }
    System.out.println(+x);
  }
}
```



Lifetime of a Variable

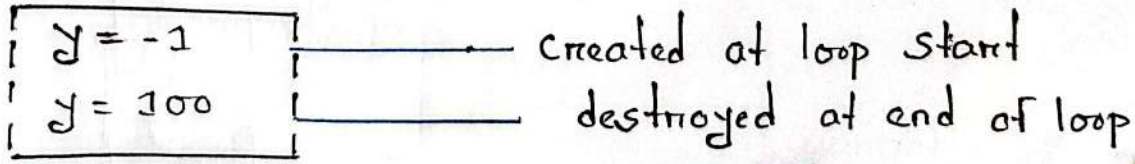
Paper Source
Subject.....
Date :.....Time:.....

```
Class Lifetime {
    public static void main (String args []) {
    int x;
    for (x=0; x<3; x++) {
        int y=-1
        System.out.println ("y is : "+y);
        y=100;
        System.out.println ("y is now:"+y);
    }
    }
}
```

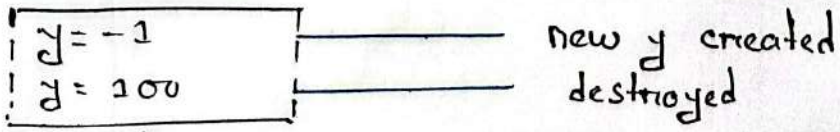
Memory / Lifetime Visualization Diagram.

(x=0; x<3; x++)

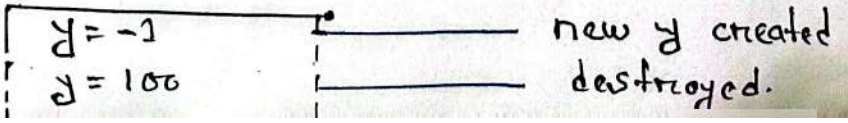
Iteration 1



Iteration 2



Iteration 3



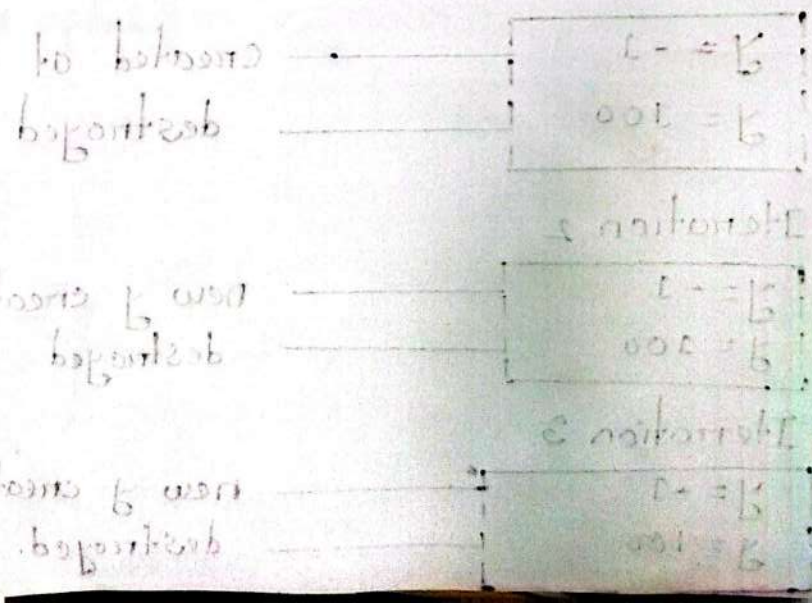
Variable name in Nested Scopes

Paper Source
Subject.....
Date :.....Time:.....

```
class ScopeErr {
  public static void main(String args[]) {
    int bar=1;
    { int bar=2; [Error]
    }
  }
}
```

Although block can be nested, a variable can not be declard to have same name as one in outer Scope.

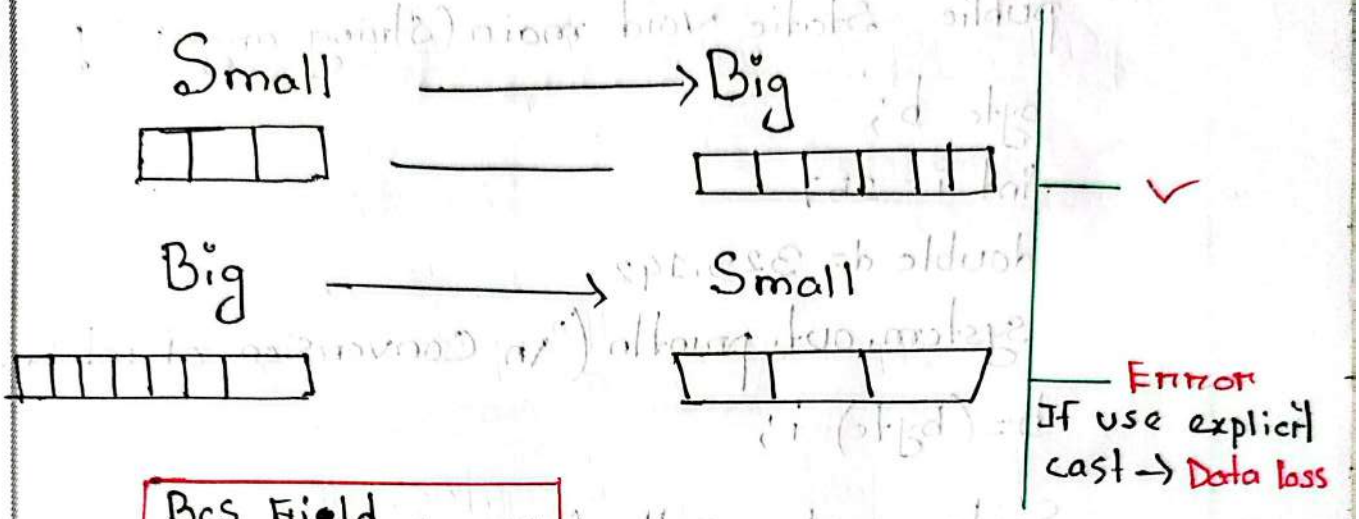
Memory:



Type conversion Compatibility

Paper Source
Subject.....
Date :.....Time:.....

Remember



BCS Field

Byte < char < Short < Float, int < long < double

1. Type convesion (compatible)

```
int a = 100
long b = a;
```

→ Automatic conversion allowed.

2. incompatible

```
double d = 10.5
byte b = d;   Error.
```

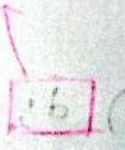
→ cannot assign directly

If convert use Explicit type casting.

```
double d = 10.5
byte b = (byte) d;
```

Paper Source
Subject.....
Date:.....Time:.....

```
Class Conversion {
    public Static void main(String args[]){
     byte b;
     int i = 257
     double d = 323.142
     System.out.println("\n Conversion of int to byte")
     b = (byte) i;
     System.out.println("i and b" + i + " " + b);
     i = (int) d;
     System.out.println("d and i" + d + " " + i);
     b = (byte) d;
     System.out.println("d and b" + d + " " + b);
    }
}
```



Automatic Type promotion

Paper Source
Subject.....
Date :.....Time:.....

```
byte b=50;
b = b*2;
```
✗ Error.

```
byte a=40;
byte b = 50;
int c = a+b
```
✓

```
byte b=50;
b = (byte) (b*2)
```
✓

byte, Short, char $\xrightarrow{\text{Promoted}}$ int

one operand long ⟶ long,

Float ——— float

double ——— double.

code

```
class promote {
    public Static void main (String args[]) {
        byte b = 42;
        char c = 'a';
        Short s = 1024;
        int i = 5000;
        float f = 5.67f
        double d = .1234;
        double result = (f*b) + (i/c) - (d*a)
    System.out.println ((f*b) + "+" (i/c) [redacted] + (d*s));
    System.out.println ("result =" + result);
```

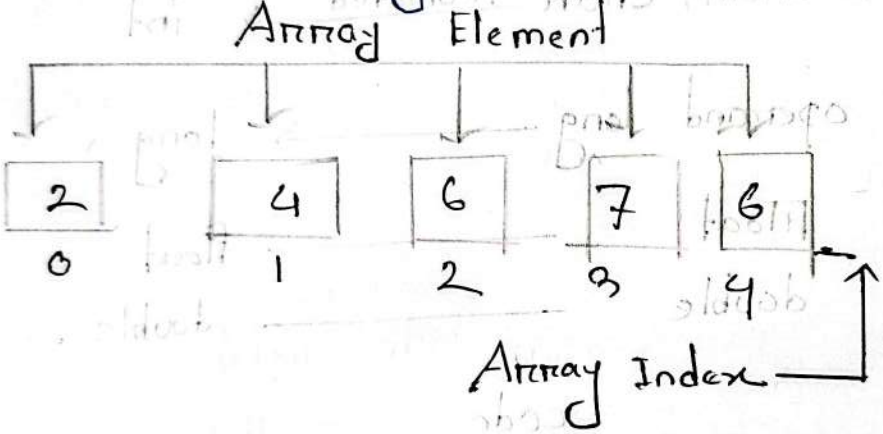
Paper Source
Subject.....
Date :.....Time:.....

Type promotion order

byte → short → int → long → float → double → Answer

↑ char

Arrays



1D array

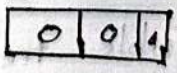
arr[] ——— object

Reference = object

int marks[] = new int[3]

Reference

new = Memory allocated for array run time

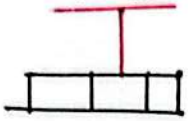


Paper Source
Subject.....
Date :.....Time:.....

1. int [] Marks = {1, 2, 3};

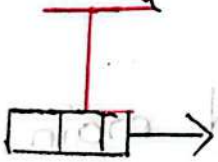
Java automaticly declar and initilize.

2. int [] marks = new int [3];



int [] marks = new int [3]; Marks -> [0, 0, 0] declaration

Marks = new int[] {85, 90, 95}; Assign



Multidimensional Array

int two [] [] = new int Column [4] [5]

[0] [0]	[0] [1]	[0] [2]	[0] [03]
[1] [0]	[1] [1]	[1] [2]	[1] [3]
[2] [0]	[2] [1]	[2] [2]	[2] [3]
[3] [0]	[3] [1]	[3] [2]	[3] [3]
[4] [0]	[4] [1]	[4] [2]	[4] [3]

Row

4 x 3

Paper Source
Subject.....
Date :.....Time:.....

Array Declaration :

① General Form

```
int a1[] = new int[3];
int[] a1 = new int[3];
```

② 2D Declaration

```
char two1[][] = new
                char[3][4];
char two2[][]
     = new char[3][4];
```

(iii) Multiple Array

```
int[] nums1, nums2, nums3;
```

```
class TwoArray {
  public static void main (String args[]){
   int TwoD[][] = new int [4], [5]
   int i, j, k = 0;

   for(i=0; i<4; i++)
     for (j=0; j<5; j++) {
       TwoD[i][j] = k;
        k++; }
   for(i=0; i<4; i++) {
    for(j=0; j<5; j++) {
      System.out.println (twoD[i][j] +" ");
   System.out println();
} }
```

Assign

Read . print .

Paper Source
Subject.....
Date :.....Time:.....

3D Array

i = 0, 1, 2 -> 3 layer
j = 0, 1, 2, 3 -> Row
k = 0, 1, 2, 3, 4 -> column

three[i][j][k] = i*j*k;
3*4*5 = 60 element

Outer loop i -> Select the entire layer.
Middle loop j -> Select Raw within layer.
Inner loop k -> Select the value column of that Row.

For each loop (Enhanced for loop)

Paper Source
Subject.....
Date :.....Time:.....

Syntax

→ object have value

```
for(dataType variable : arrayorcollection) {
```

Temporary shortage

* The variable data type must be compatible arrays element

```
int number[] = {10, 20, 30, 40}
for (int n : number) {
    System.out.println(n);
```

Array object have value

→ Diagram

Array	10	20	30	40
Index	0	1	2	3

n = number[0] = 10
print(10)

Next n = number[1] = 20
print(20

Paper Source
Subject.....
Date :.....Time:.....

Summing element in a 1D array

```
public class ForEach1D {
    public static void main (String [] args) {
        int [] marks = {85, 90, 78, 92, 88};
        int Sum = 0;
        for (int Score : Marks) {
            Sum += Score;
        }
        System.out.println ("Total Marks:" + Sum);
        System.out.println ("Avarage Marks:" + (sum / marks.length));
```

0 1 2 3 4

4

85	90	78	92	88
0	1	2	3	4

Score = 85 → Sum = 85
Score = 90 → Sum 85+90 = 175.

Inner loop pic Value

Index		
0	Row0[1 2 3]	0 0 0
1	Row1[4 5 6]	0 0 0
2	Row2[7 8 9]	0 0 0

Class.

Paper Source
Subject.....
Date :.....Time:.....

```
Class classname{
  type instance-variable 1;
  type instance-variable 2;
  type instance-variable N;

  type methodname1(parameter-list){
  }type methodname2(parameter-list){
  }
  type methodnameN(parameter-list){
  }
}
```

Type = Data type
instance variable
= obj property fie

Method name = name of the Method, like setdetails add,

parameter = input variable of Method.

```
class Box{
  int width, height, depth;
}
public class main{String
```

class → Design only (no memory)
new → creates real obj (memory allocated).

Mybox1

Width = 100
height = 50
depth = 30

access by mybox1.width
mybox1.height
mybox1.depth

***

object + dot + variable = data access

```
class Box{
    double width;
    double height;
    double depth;
}

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

        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;

        mybox2.width = 3
        mybox2.height = 6.
        mybox2.depth = 9

    vol = mybox1.width * mybox1.height * mybox1.depth
    System.out.println("volume of Box1" +vol);

    vol = mybox2.width * mybox2.height * mybox2.depth
    System.out.println(mybox2+vol);
}
```

Visu

Paper Source
Subject.....
Date :.....Time:.....

Class Template:

Box
- width
- height
- depth

Runner class

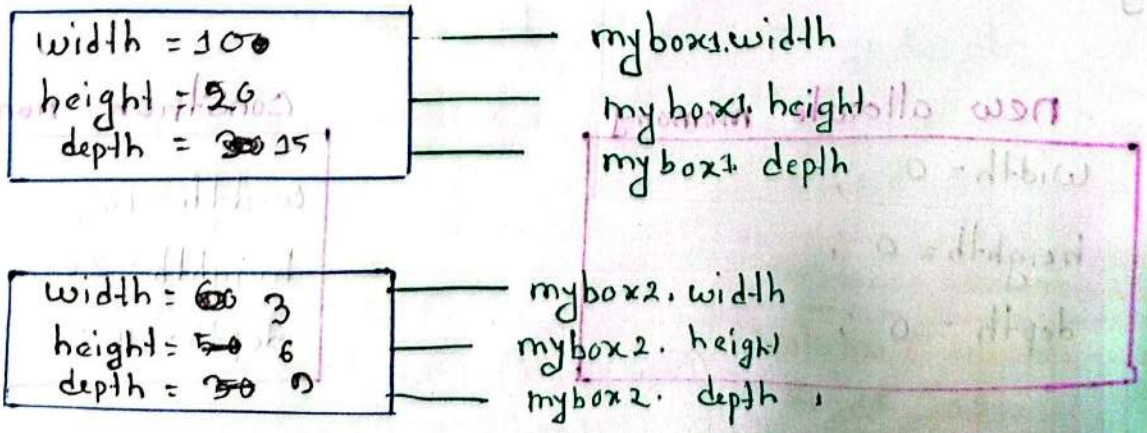
BoxDemo
- main()
  - Creates mybox1 -> newbox
  - Creates mybox2 -> newbox
  - Calculated and prints

file name = main method name

Heap Memory (objects creates by (new))

mybox1 -> [width=0.0 | heighth=0.0 | depth=0.0]

mybox2 -> [width=0.0 | heighth=0.0 | depth=0.0]



Result calculate

Constructor

Paper Source
Subject.....
Date :.....Time:.....

```
class Box {
    double width;
    double height;
    double depth;

    Box() {
        width = 10;
        height = 20;
        depth = 15;
    }

class BoxDemo {
    public static void main (String [] args) {
        Box mybox = new Box();
    }
}
```

constructor call ()

constructor = Same name as class
no return type

constructor call ()

new allocate memory

```
width = 0;
height = 0;
depth = 0;
```

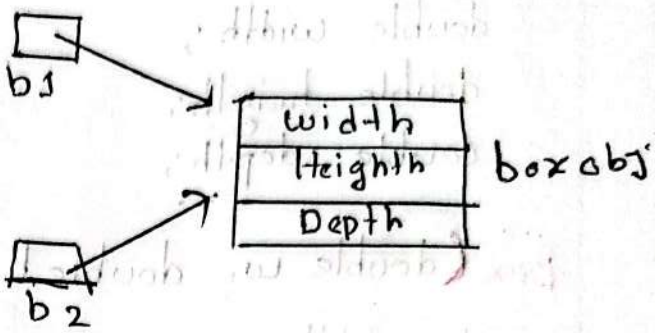
Constructor run automatically

```
width = 10;
height = 20;
depth = 15;
```

Void

- read obj data
- perform action (calculation, print, update etc
- returns = nothing

output change but no value return;



box b1 = new Box();
box b2 = b1;

```
class Box {
  double width;
  double height;
  double depth;

  double volume() {
    return width * height * depth;
  }

  void setDim(double w, double h,
              double d) {
    width = w;
    height = h;
    depth = d;
  }
}

class BoxDemo5 {
public static void main(String args[])
{  Box mybox = new box1();
   mybox.setDim(10,20,30);
 vol = mybox.volume
```

parameters

arg

width = ?
heigth = ?
depth = ?

setDim(double w, h, d) - parameter

w = 10
h = 20
d = 15

- Assign to obj

width = w
heigth = h
depth = d

Instance Variable updated.

Volume() method
- Reads instance vars
width = 10, depth = 30
heigth = 15

calculate: 3000

parameter Constructor

Paper Source
Subject.....
Date :.....Time:.....

```
class Box {
    double width;
    double height;
    double depth;

Box (double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
    double volume() {
        return width * height * depth;
}

class BoxDemo {
    public static void main (String args[]) {
        Box mybox1 = new Box(10,20,15);
        System.out.println(mybox1.volume());
```

Paper Source
Subject.....
Date :.....Time:.....

this. → instance Variable Hiding

```
width = width X  Error
this.width = width
```

or width = w

Null

```
Box mybox = new Box();  ——→ create obj.
mybox = null;           ——→ remove reference.
```

Finalize

last chance clean up

```
class Box {
  protected void finalize() {
    System.out.println("Box destroy");
```

Stak

Paper Source
Subject.....
Date :.....Time:.....

Index	Value
1	
2	
3	
4	
5	

Top-1,

Index	value
0	10 ← top
1	
2	
3	
4	
5	

Top-0

Index	Value
0	10
1	20 top
2	30
3	40
4	50

top 1 Top 2 Top 4 ?

pop

Index	Value
0	10
1	20
2	30
3	40
4	50

pop = Top-4
= 1.

Paper Source
Subject.....
Date :.....Time:.....

Method Overloading

moobj → Method overloading obj

- add (int a, int b)
  a= , b=
  return ab value
- add (double a, double b)
  a=value be value
  return value

```
class MethodOverloading {
    public int add(int a, int b)
        return a+b; }
    public double add ( double a, double b){
        return a+b; }
}
public class MethodOverloadingDemo {
        Methodoverloading mobj = new methodover
    int Sum1 = Mobj.add(5,10);
    System.out.println("Sum of two double:"+sum1);
```

Paper Source
Subject
Date Time

Overloading constructor

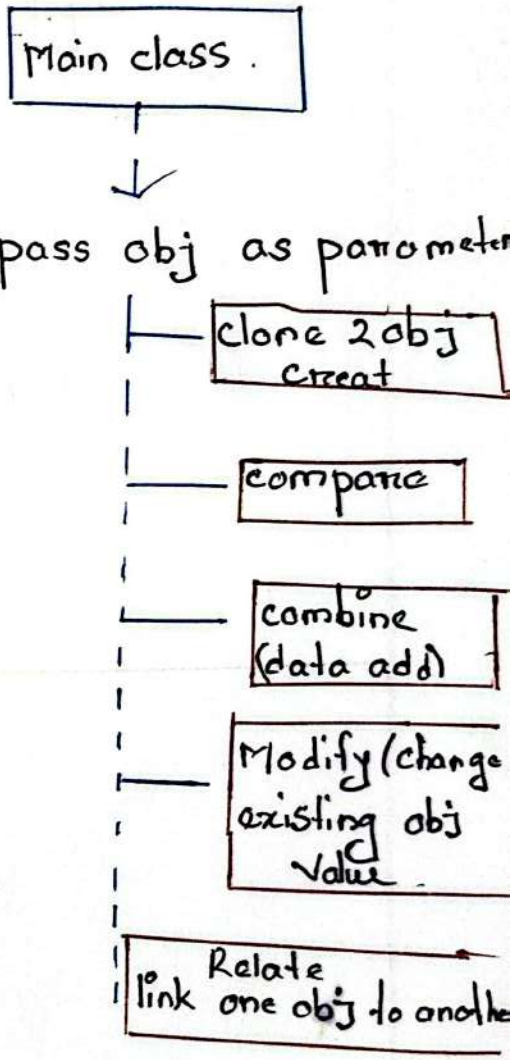
Conceptual Note

- Same class but multiple constructor different type & parameter
- obj create, Java use parameter = call

Paper Source
Subject.....
Date:.....Time:.....

Using obj as a parameter

```
class test {
 int a,b;
Test (int i, int j) {
  a = j
  b = j
}

boolean equalto (Test o) {
 If (o.a == a && o.b == b)
   return true;
else
   return false;
}
}
class passob {
public static void main (String s
                        args[])
 Test ob1 = new test (100, 22)
 Test ob2 = new test (100.22)
 Test ob3 = new test (-1, -1);
System.out.println("ob1 == ob2:"
                   +ob1.equalto(ob2));
System.out.println("ob1 == ob:3"
                   +ob1.equalto (ob3))
```



ob1 -> [a=100, b=22]
ob2 -> [a=100, b=22]
ob3 -> [a=-1, b=-1]
└ reference pointing memory

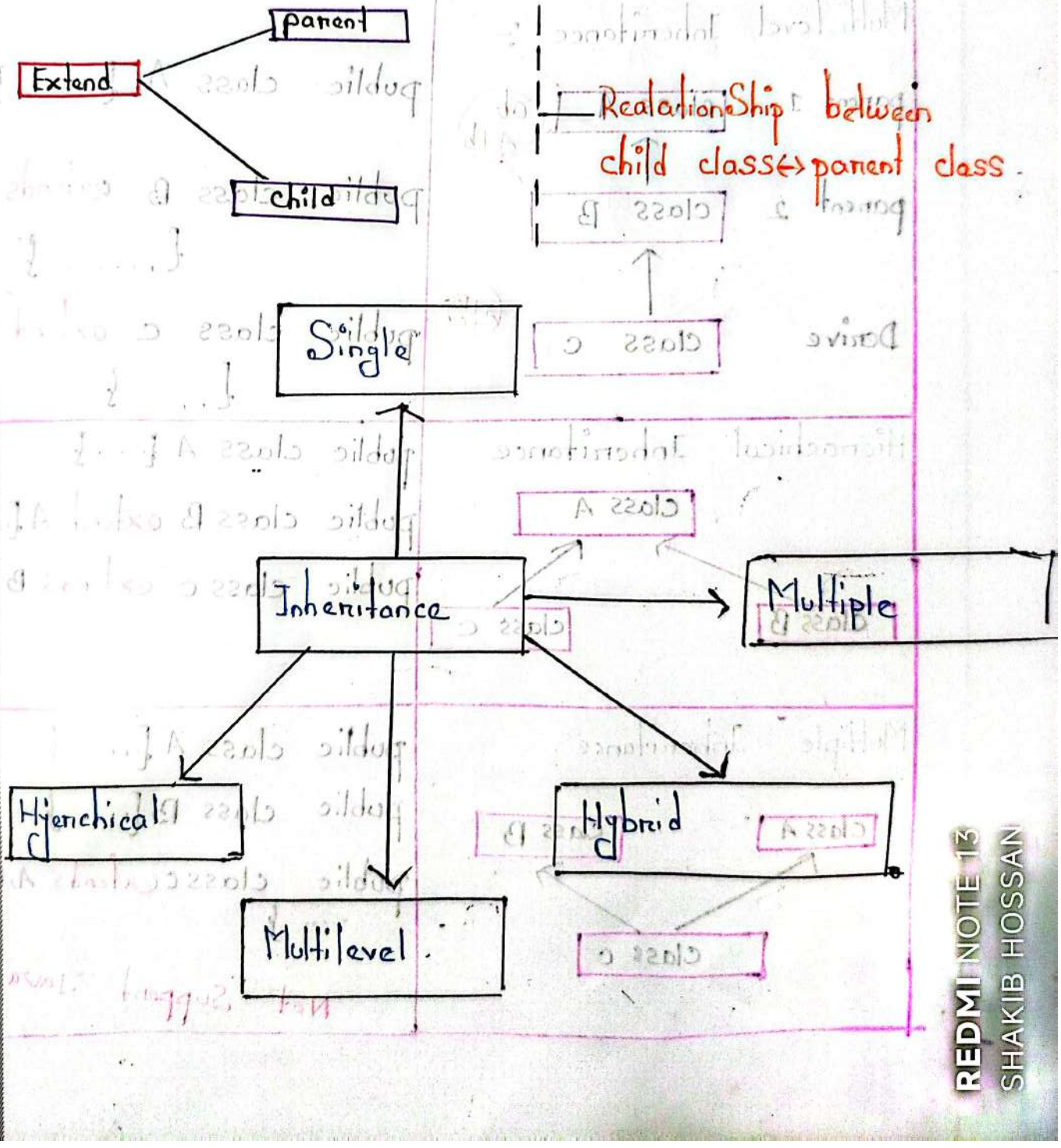
This=ob1 | compare
O = ob2 | condition

Inheritance

Paper Source
Subject.....
Date :.....Time:.....

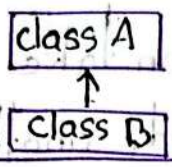
Inheritance Basics

Inheritance :- One class can take (reuse) the properties Method another class.



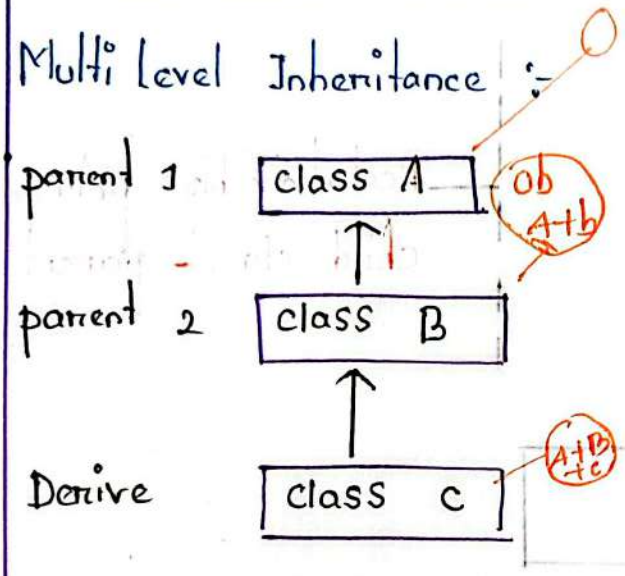
Paper Source
Subject.....
Date :.....Time:.....

Single Inheritance :-



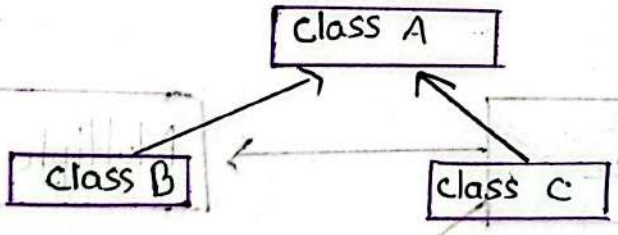

```
public class A{
.....
}public class B extends A{
}
```

Multi level Inheritance :-



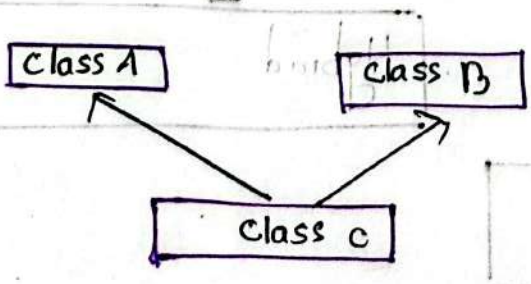

```
public class A {.....}

public class B extends A
        {.....}

public class C extends B
        {.....}
```

Hierachical Inheritance




```
public class A {.....}
public class B extends A{...}
public class C extends B{...}
```

Multiple Inheritance



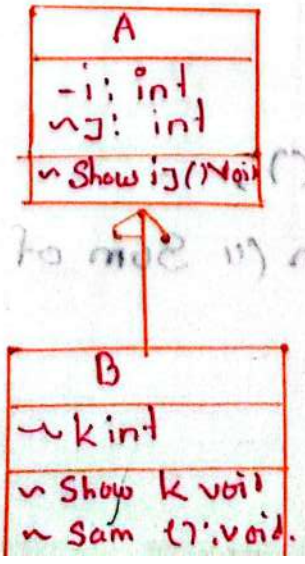

```
public class A {.....}
public class B{.....}
public class C extends A,B{
...}
```

Not Support Java.

Paper Source
Subject.....
Date :.....Time:.....

```
class A {
int i,j;

Void  Show () {
   System. out. println("i and j:"+i+" "+j);
 }
}

class B extends A{
  int k;

Void  Showk () {
    System. out. println("k:"+k);

Void  Sum () {
 System. out. println("i+j+k: "+(i+j+k));
  }
}
```

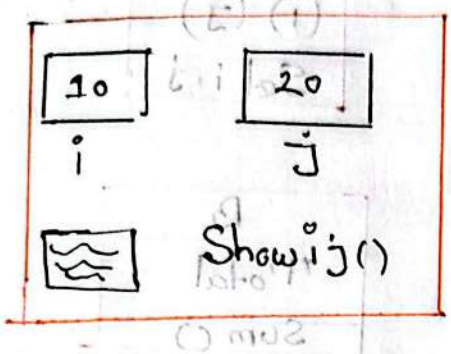
class A → Superclass.
class B → Subclass.

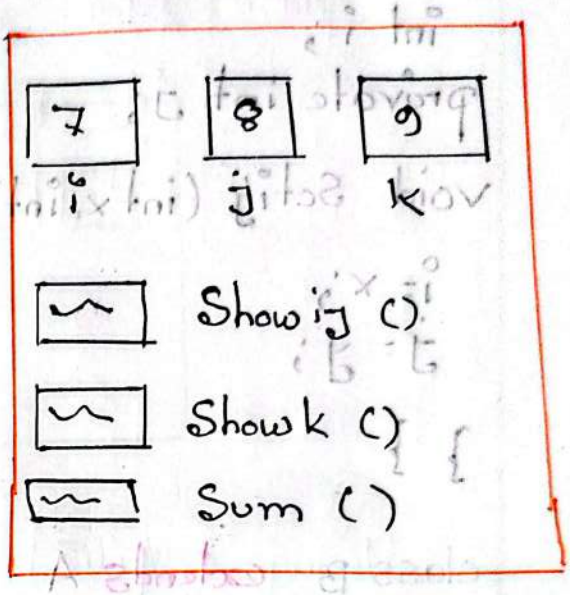


Paper Source
Subject
Date Time

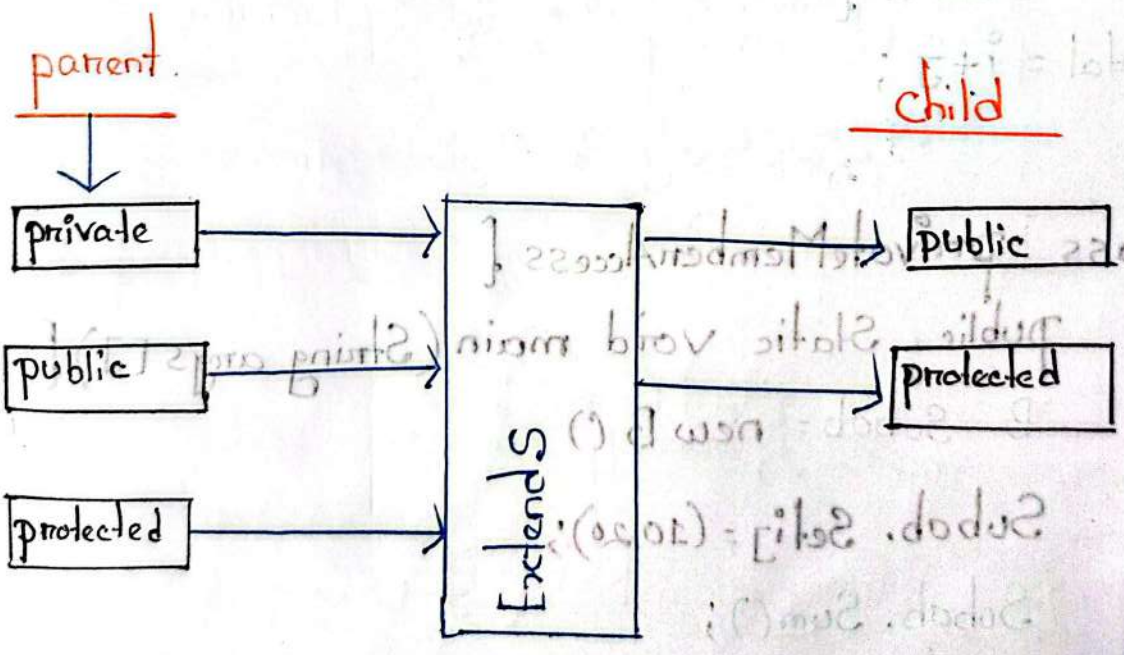
```
class SimpleInheritance {
    public static void main (String args[]) {
        A Superob = new A ();
        B Subob = new B ();

        Superob.i = 10;
        Superob.j = 20;
        System.out.println (" contents of Superob: ");
        Superob.showij ();
        System.out.println ();

        Subob.i = 7;
        Subob.j = 8;
        Subob.k = 9;
        System.out.println (" contents of Subob: ");
        Subob.showij ();
        Subob.showk ();
        System.out.println ();
        System.out.println (" Sum of i, j and k in Subob: ");
        Subob.sum ();
```

Paper Source
Subject.....
Date :.....Time:.....



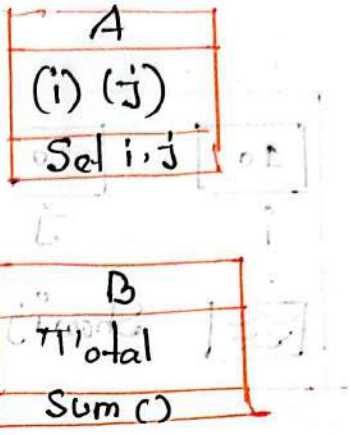


Member Access and Inheritance



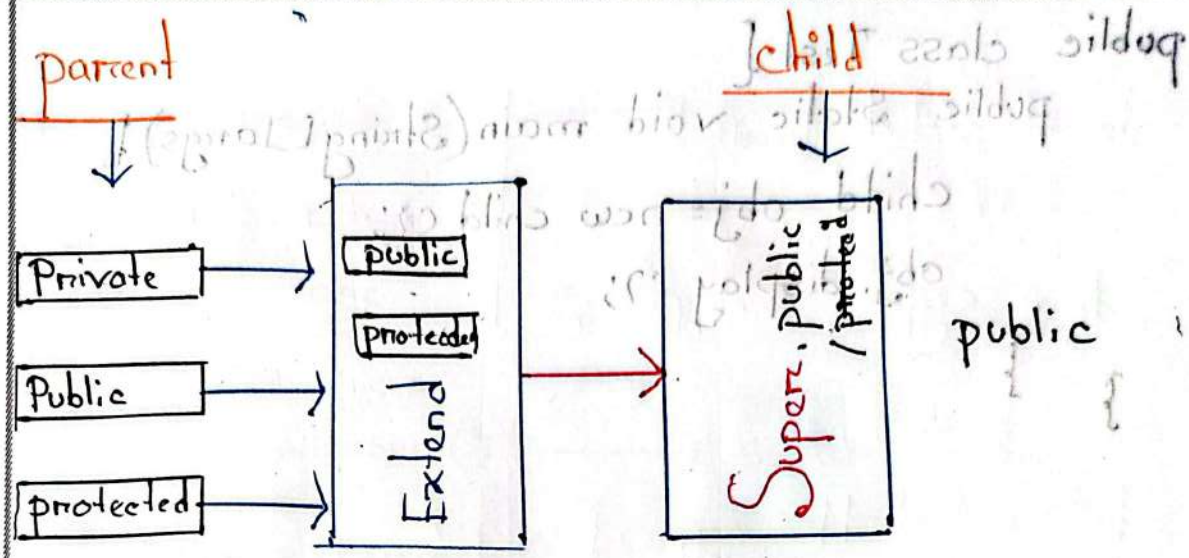
Paper Source
Subject.....
Date :.....Time:.....

```
class A {
 int i;
private int j;
void setij (int x, int y) {
  i = x;
  j = y;
} }

class B extends A
 int total;
void Sum () {
 total = i+j;
} }

class privateMemberAccess {
    public Static void main (String args []){
     B Subob = new B ()
     Subob. setij = (10,20);
     Subob. Sum();
     System.out.println (" Total is "+ Subob. total);
```



Super Key Concept

Paper Source
Subject.....
Date :.....Time:.....



Example

```
Class parrent {
   int x=10; int y=30; int z=20;
 Void Show(){
   System.out.println("parent x="+x);
    }
}
class child extends parent {
    int x=20;
 Void display() {
    System.out.println("child x="+x);
    System.out.println("Parent x="+ Super.x);
    Super.Show();
} }
```

Paper Source
Subject.....
Date :.....Time:.....

```
public class Test {
    public Static void main(String[]args) {
        child obj= new child ();
        obj. display ();
    }
}
```

Using Super To Call Superclass Constructor.

```
class person {                          ——— parent class
    private String name;
    private int age;

    person () {                         ——— Default constructor
        name = "Unknown";
        age = -1
    }
    person (String n, int a) {          ——— parameterized constructor
        name = n;
        age = a;
    }
}
```

Paper Source
Subject.....
Date :.....Time:.....

```
person(person p) {          ⟶ clone constructor
name = p.name
age = p.age

Void Showperson() {         ⟶ getter Method
System.out.println("Name=" + name);
System.out.println("Age=" + age);
}}
```

```
class Student extends person {
    private double cgpa;
//Default constructor,
    Student() {
        super()             ⟶ calls parent class
        cgpa = -1;
```

3rd obj
S3

```
Student(String name, int age, double cgpa) {
// parameter Constructor
    super(name, age);       ⟶ calls parent class two variable
    this.cgpa = cgpa;       ⟶ just child class
```

S

```
Student (Student s1) {
    Super (s);
    this. cgpa = s. cgpa ;
}
```

→ S4 obj

```
Student (double cgpa) {
    Super ();
    this. cgpa = cgpa;
}
```

→ S2 obj

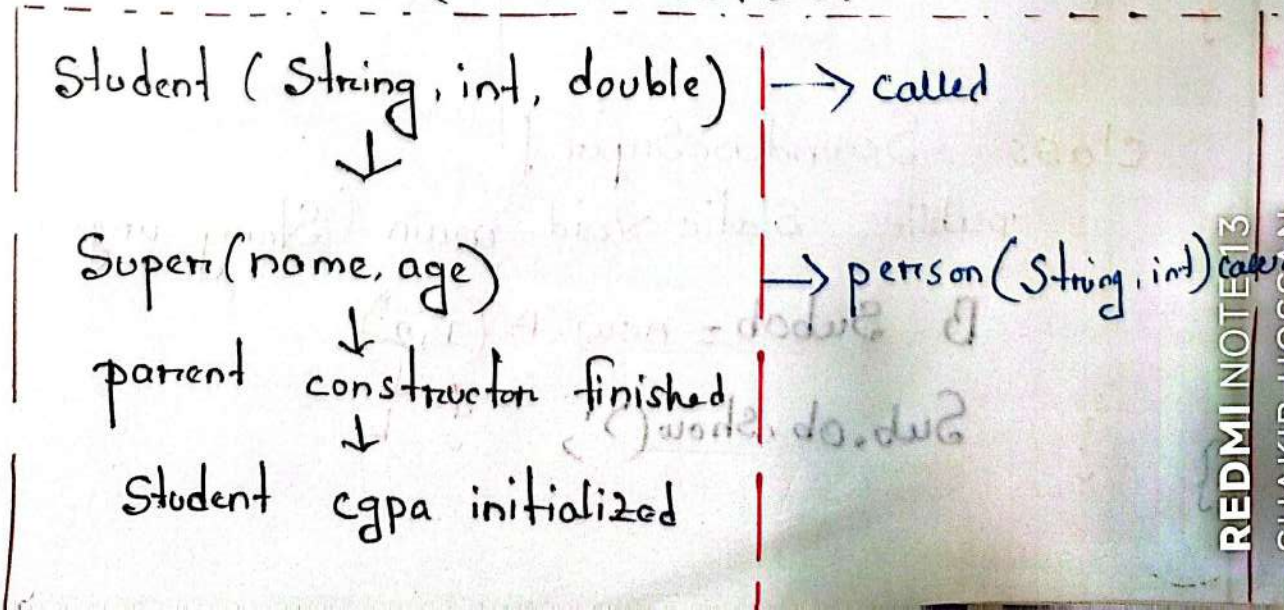
```
Void    ShowStudent () {
          Showperson ();
          System. out. println ("cgpa =" +cgpa);
        }
}
```

Date :..... Time:.....

```
class DemoSuper {
    public Static void main (String [] args) {

        Student s1 = new Student (" Shakib", 20. 3.84);
        Student s2 = new Student (3.50);
        Student s3 = new Student ();
        Student s4 = new student (s1);

        S1. ShowStudent ();
        S2. ShowStudent ();
        S3. ShowStudent ();
        S4. Show Student ();
```

Flow Diagram

S1 = new student ("Shakib", 20, 3.84)

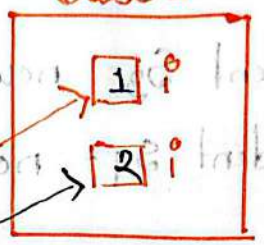


Paper Source
Subject.....
Date :.....Time:.....

Second Use of Super.

```
class A {
    int i;
}

class B extends A {
    int i;
    B(int a, int b) {
        super.i = a;
        i = b;
    }
    void show() {
        System.out.println("i in the Superclass:" + super.i);
        System.out.println("i in the Subclass:" + i);
    }
}

class SecondUseSuper {
    public static void main(String args[]) {
        B subob = new B(1,2);
        sub.ob.show();
    }
}
```

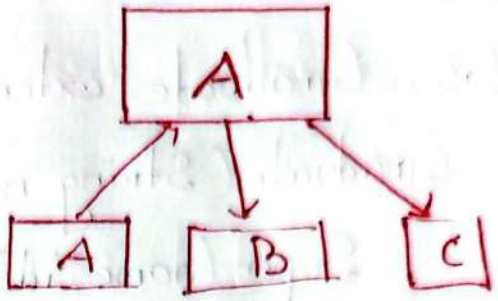
Subclass member name = Superclass member name



(1) (2) (a) (b)

Paper Source
Subject.....
Date:.....Time:.....

Three levels of Inheritance Hierarchy




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

    Student ( String name, int id) {
        this.name = name;
        this. Id = id;
    }
    void Study() {
        System.out.println(name + " is Studing.");
    }
}
```

#| child 1

```
class Undergraduate extends Student {
    Undergraduade ( String name, int id) {
        Super(name, id);
    }
    Void projectwork() {
        System.out.println(name + " is doing undergraduate
                                    project.");
    }
}
```

#Child 2

```
class Graduate extends Student {
    Graduate(String name, int id) {
        Super(name, id);
    }
    void thework() {
        System.out.println(name + " is doing master ");
    }
}
```

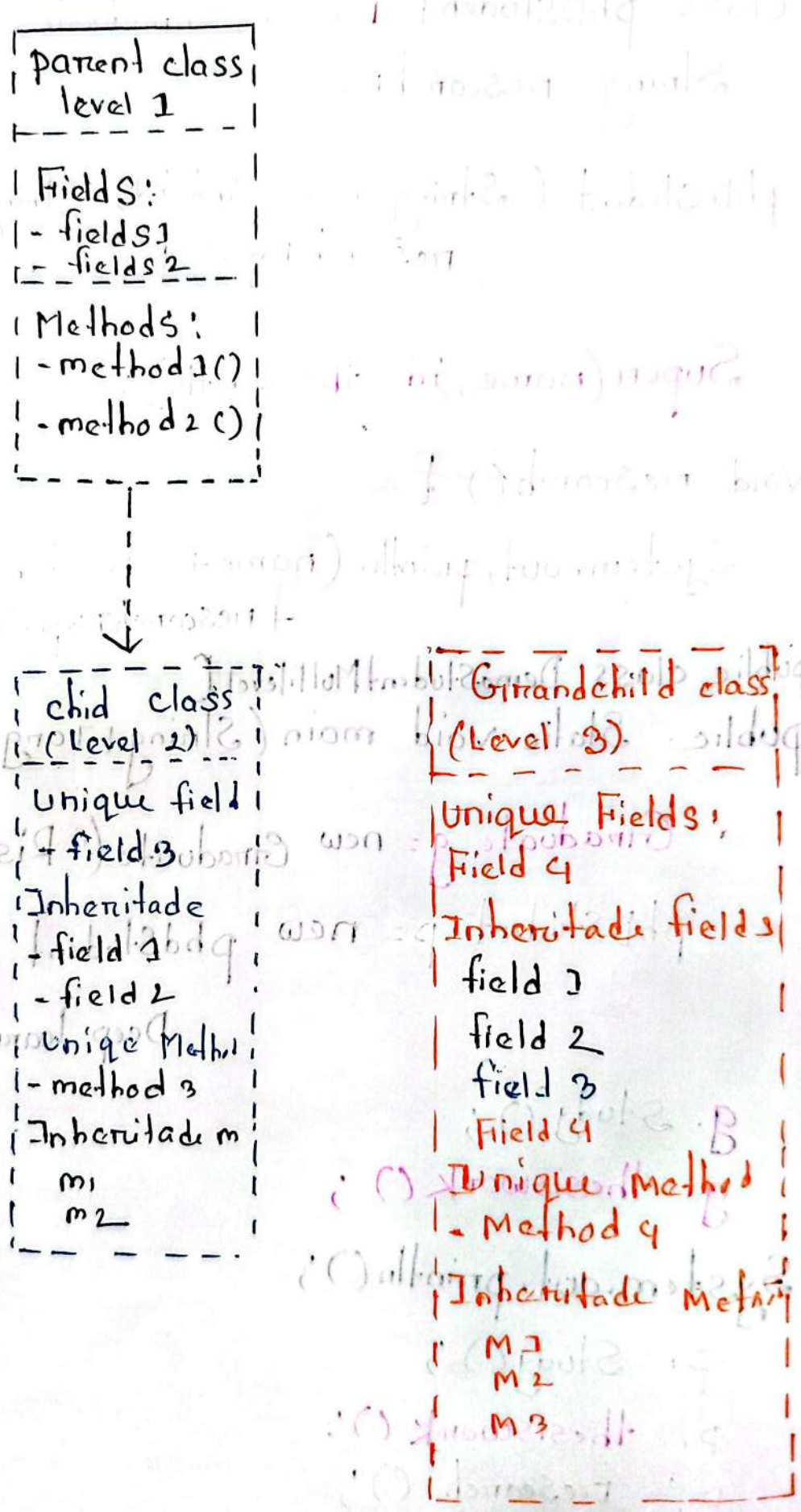
```
class phDStudent extends Student {
    phDStudent(String name, int id) {
        Super(name, id);
    }
    Void research() {
        System.out.println(name + " is doing phd.");
    }
}
```

Paper Source
Subject.....
Date :.....Time:.....

```
public class DemoStudentHierarchy {
    public Static Void main (String [] args) {

        Undergraduate u = new Undergraduates ("Arisha", 101);
        Graduate g = new graduates ("Risha", 201);
        phDStudent p = new phDstudent ("Diya", 301);

        u.Study();
        u. projectwork();

        g. Study();
        g. thesiswork();

        p. Study();
        p. Research

    }
}
```

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

    Student (String name, int id) {
    this.name = name;
    this.Id = id;
    }
  void Study () {
      System.out.println(name+" is Studing.");
    }
}
class Graduate extends Student
 String thesisTitle;
Graduate (String name, int id, String thesisTitle) {
    Super (name, id);
 this.thesisTitle = thesisTitle;
 }
void thesiswork() {
      System.out.println (name + "is doing mast'
                          + thesisTitle)
}
```

```
class phDStudent Extends Graduates {
  String researchTopic;

phDStudent ( String name, int id, String thesisTitle, String
              researchTopic) {

  Super(name, id, thesisTitle);
Void research() {
  System.out.println(name+" is doing phD research on!!
                      +researchTopic);
public class DemoStudentMultilevel{
public Static void main(String[]args) {

    Graduate g = new Graduate("Risha", 201, "AI in
                               Education");
    phDStudent p = new phdStudent("Diya", 301, "
                    Deep learning optimization");

  g.study();
  g.thesiswork();
System.out.println();
  P. Study();
  P. thesiswork();
  P, research();
```

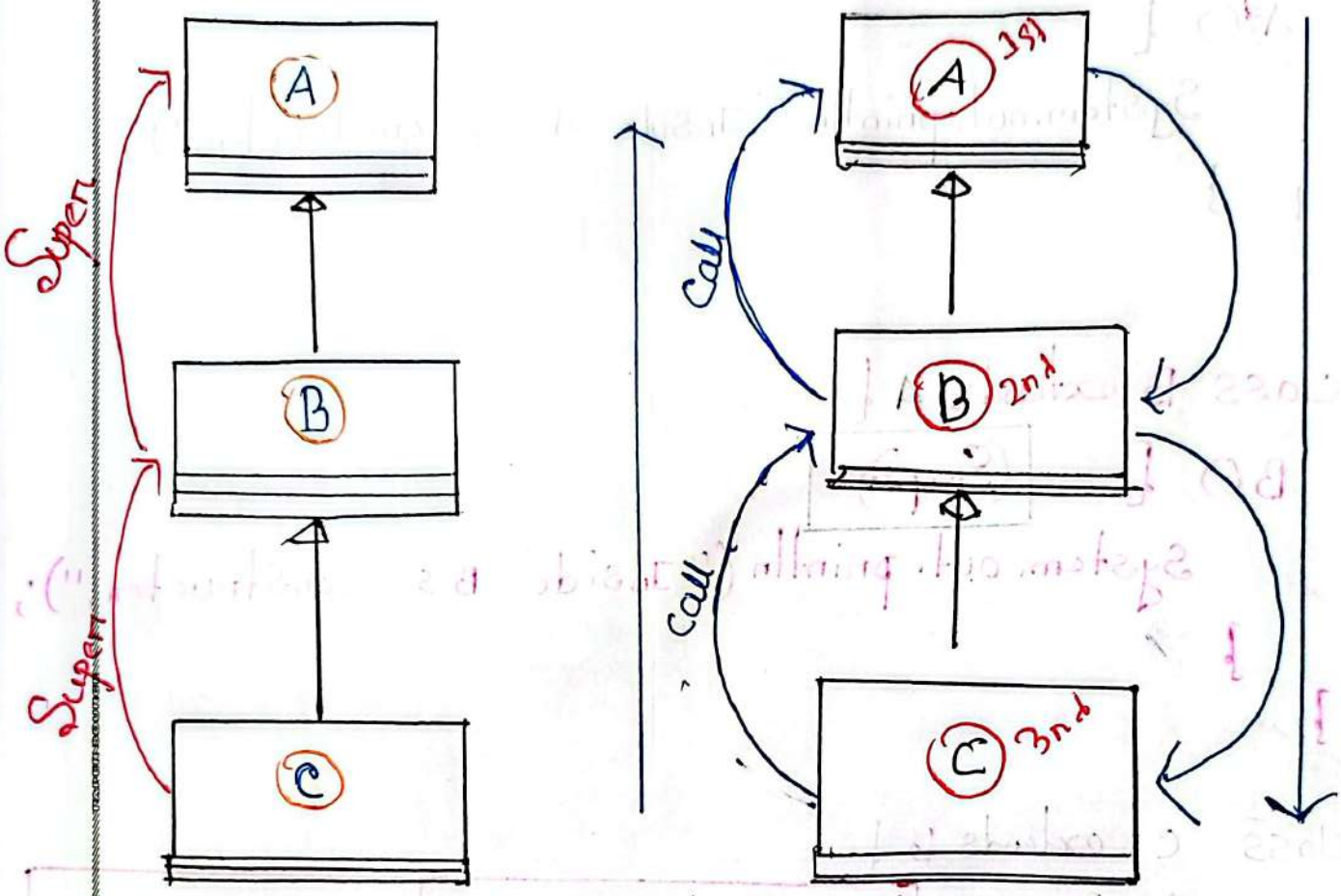

parent class
level 1
Fields:
- fields 1
- fields 2
Methods:
- method 1()
- method 2 ()
child class
(level 2)
Unique field
+ field 3
Inheritade
- field 1
- field 2
Unique Meth
- method 3
Inheritade m
m1
m2
Grandchild class
(Level 3)
Unique Fields:
Field 4
Inheritade fields
field 1
field 2
field 3
Field 4
Unique Method
- Method 4
Inheritade Meth
M1
M2
M3

Constractor Call Sequence

Paper Source
Subject.....
Date :.....Time:.....

```
class A {
  A () {
    System.out.println ("Inside A's constructor,");
  } }

class B extends A {
  B() {  (Super)
    System.out.println ("Inside B's constructor,");
  }
}

class C extends B {
  C ()  {  ⟶ (Super) is called when no constructor written explicitly
    System.out.println ("Inside C's constructor,");
  }
}

class callingcons {
    public static void main (String args []) {
        C. c = new c();
    }
}
```

call



1st Variable initilaization, 1s Variable Depand another.

Method Overriding

Paper Source
Subject.....
Date :.....Time:.....

Overloaded -> Name Same but parameter not same

```
Class A{
int i, j;
A(int a, int b)
  i = a;
  j = b;
void Show(){
   System.out.println("i and j:"+i+" "+j);
  }
}

class B extends A{
   int k;
B(int a, int b, int c){
   Super(a,b);
   k = c;
}
void Show(String msg){
System.out.println(msg+k);
  }
}

class OverloadNotOverride{
public static void main(String args[]){
B Subob = new(1,2,3);
Subob.Show("This is k:");
```

empty

Sub.Show();

not empty

// Show in B

Subject.....
Date :.....Time

```
class A {
 int i,j;

A(int a, int b) {
  i = a;
  j = b;
}
Void Show() {  ⟶ Same name, parameter Same,
    System.out.println("i and j:"+i+" "+j);
} }

class B extends A {
   int k;
 B(int a, int b, int c) {
    Super(a,b);
    k = c
 }
  Void Show() {  ⟶ Same name, parameter Same
      System.out.println("k:"+k);
   }
}
  class Override {
      public static void main(String args[]) {
calling obj  B A Subob = new B(1,2,3), A(1,2)
depend
      Subob.Show();
```

Dynamic Method Depth

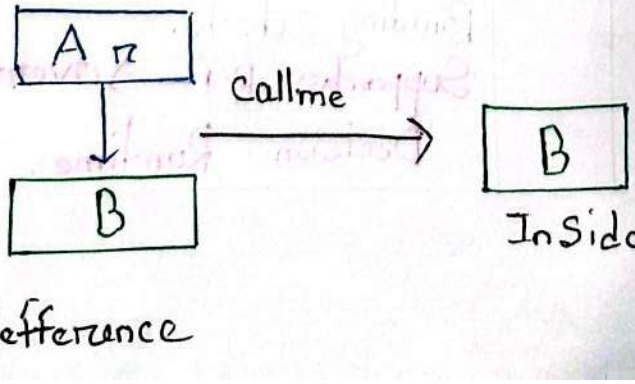
Dynamic Method Depth

Dynamic Method Depth

Dynamic Method Depth

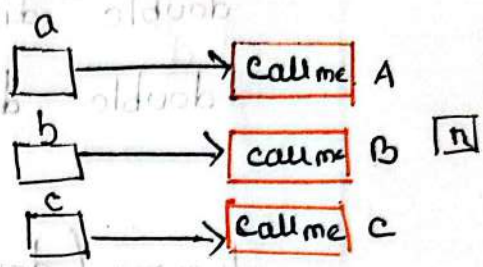
Dynamic Method Dispatch Example.

```
class A {
    void callme() {          ⟵ Overridding
        System.out.println("Inside A's call
    }
}

class B extends A {          ⟵ Inheritance check
    void callme() {          ⟵ Overridding ditect
        System.out.println("Inside B's callmet
    }
}

class C extends A {
    void callme() {          ⟵ Overridding ditecti
        System.out.println("Inside C's callme
    }
}
```



Paper Source
Subject.....
Date :.....Time:.....

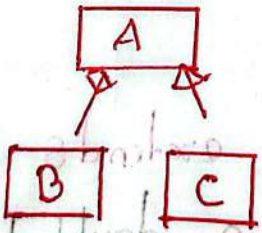
```
class Dispatch {
    public static void main (String args[]) {
        A a = new A();
        B b = new B();
        C c = new c();
```

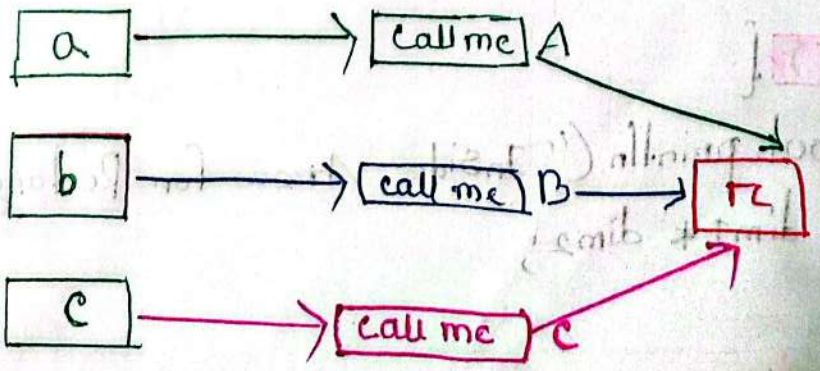



```
A π;  ──→ upcasting
    π = a;
    π. callme();
    π = b;
    π. callme();
    π = c;
    π. callme();
```

→ Binding check.

```
    }
}
```





→ Dynamic Dispatch

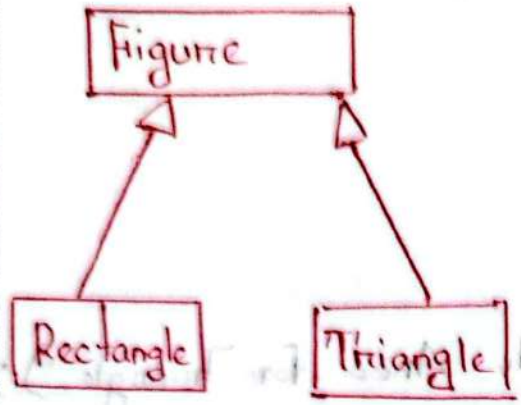
Method Overriding Applying

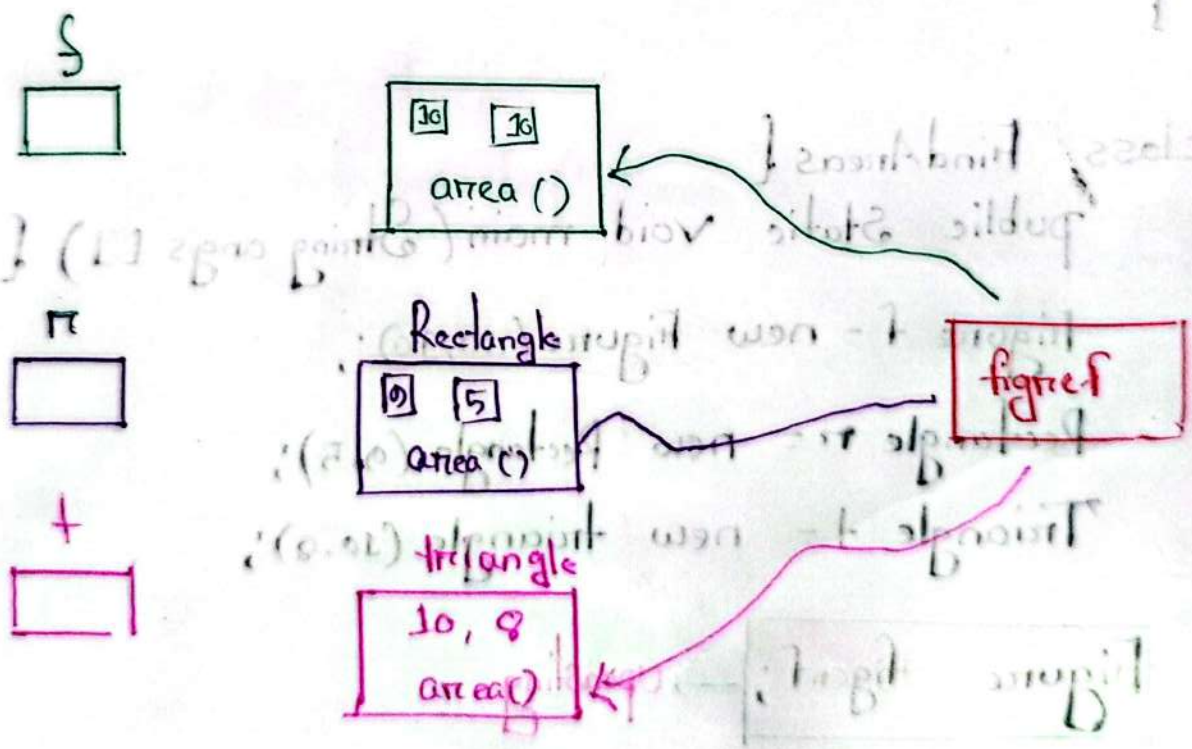
```
class Figure{
    double dim1;
    double dim2;

    Figure(double a, double b){
        dim1 = a;
        dim2= b;
    }

    double area(){
        System.out.println("Area For Figure is undefin
        return 0;
    }
}

class Rectangle extends Figure{
    rectangle(double a, double b){
        super(a,b);

double area(){
    System.out.println("Inside Area for Rectangle:");
    return dim1 * dim2;
    }
}
```

```
class Triangle extends Figure {
    Triangle (double a, double b) {
        Super (a, b);
    }

    double area () {
        System.out.println (" Inside Area for Triangle.");
        return dim1 * dim2 / 2;
    }
}

class FindAreas {
    public Static void main ( String args [] ) {
    Figure f = new Figure (10, 10);
    Rectangle r = new Rectangle (9, 5);
    Triangle t = new triangle (10, 8);

    Figure figref;  ---> Upcasting
    figref = r;
    System.out.println ("Area is" + figref.area ());
    figref = t;
    System.out.println ("Area is" + figref.area ());
    figref = f;
    System.out.println ("Area is" + figref.area ());
```

Paper Source
Subject.....
Date :.....Time:

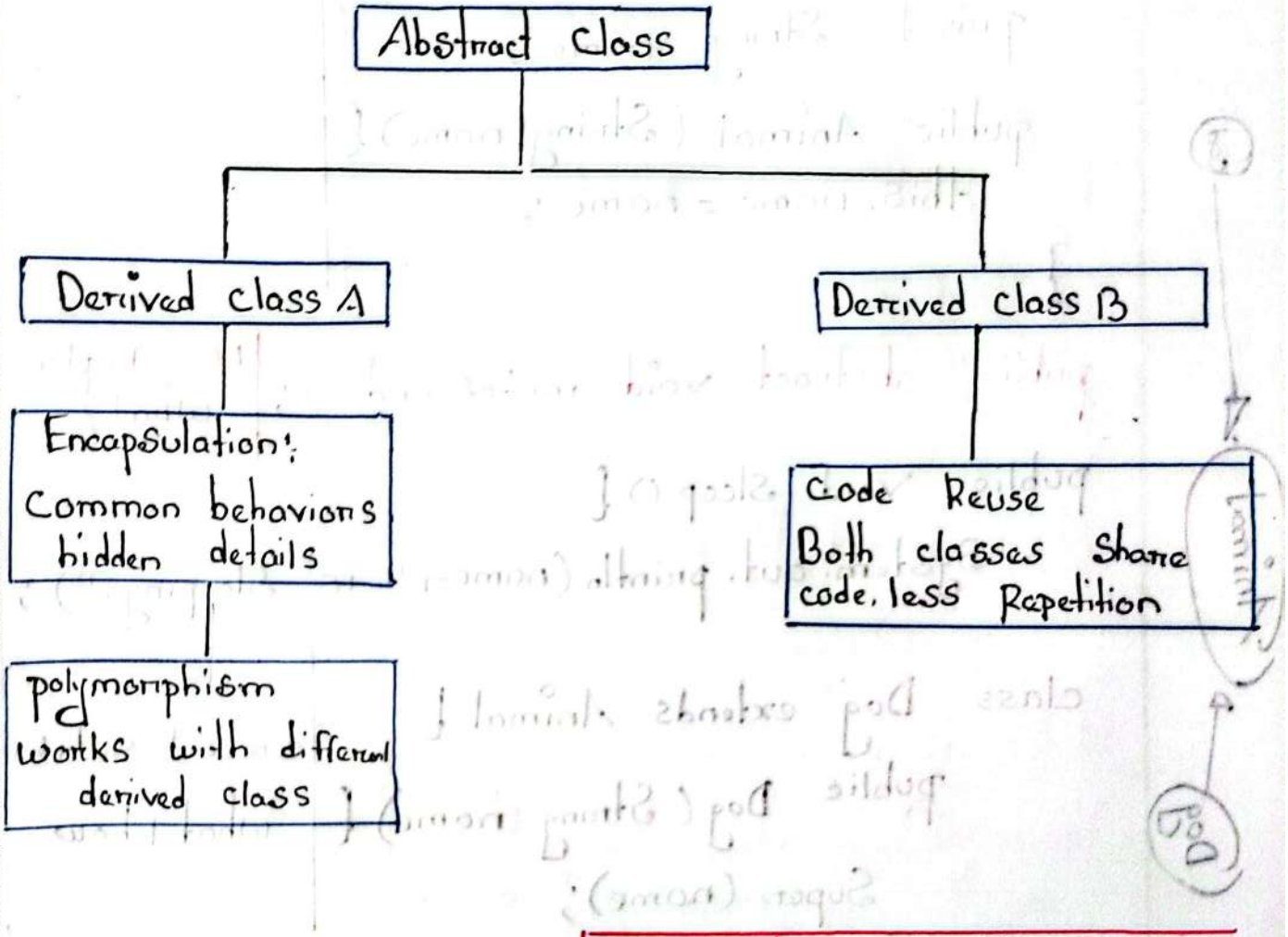
UML





Abstract Classes

Subject.....
Date :.....Time:.....



Use the abstract keyword

contains abstract method (without body)

object creation is not allowed

Subclass Must implement all abstract methods.

Abstract method has no Implementation body, and each Subclass must provide its own Implementation body

Paper Source
Subject.....
Date :.....Time

Abstract class Example.

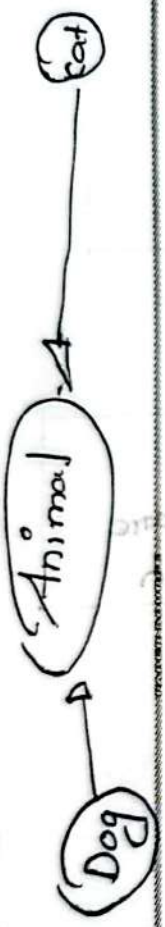
```
abstract class Animal {
    private String name;
    public Animal (String name) {
        this.name = name;
    }
    public abstract void makeSound ();
    public void Sleep () {
        System.out.println (name + " is Sleeping.");
    }
}
class Dog extends Animal {
    public Dog ( String name) {
        Super (name);
    }
    public void makeSound () {
        System.out.println (" Dog barks!");
    }
}
```

Abstract defina

No Implemen what?

concreat Subcla what + how

implementa abstract



```
class Cat extends Animal {          // concreat Subclass
    public cat ( String name) {
        Super (name);
    }

public void makeSound () {          // Implemention body
    System.out. println (" Cat meows!") ;

public class AbstractAnimal {
  public Static void main ( String args [] ) {

    Dog dog = new Dog ("Tom");
    Cat cat = new cat ("whiskers");

    dog. makeSound ();
    dog. Sleep ();

    cat. makeSound ();
    cat. Sleep ();
  }
}
```

```
abstract class Figure{
    double dim1;
    double dim2;

  Figure(double a, double b){
     dim1 = a;
     dim2 = b;
  }

  abstract double area();

}
class Rectangle extends Figure{
 Rectangle (double a, double b) {
      Super (a,b);
  }

 double area(){
     System.out.println(" Inside Area for Rectangle");
     return dim1 * dim2;
    }

}
```

```
Class Triangle extends Figure {
    Triangle (double a, double b) {
        Super (a, b);
    }

    double area () {
        System. out. println (" Inside Area for triangle.");
        return dim* dim2 /2;
    }
}

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

    Rectangle r = new Rectangle (9,5);
    Triangle t = new Triangle (10, 8)

    Figure figref;

    figref = r;
    System. out. println ("Area is" + figref. area ());
    figref = t;
    System. out. println ("Area is" + figref. area ());
```