Object-Oriented Programming

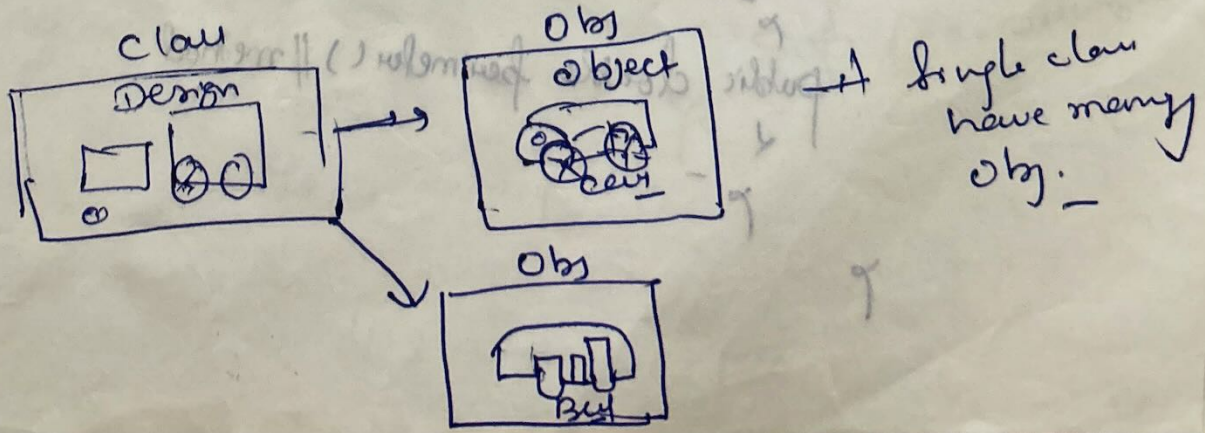
Principal of OOP's:

- Abstraction
- Encapsulation
- Inheritance
- polymorphism

- **Abstraction** means hidden internal detail and showing only require feature like function
- **Encapsulation** :- The binding of member variable and member function into a single Component.
- **Inheritance** :- The derived class is taking all the function of parent class and adding some extra feature. (specialization)
- **polymorphism** :- (Generalization) Binding similar type of task groups.
  like → Student, Traffic light
  stop all Car.

Class - Object

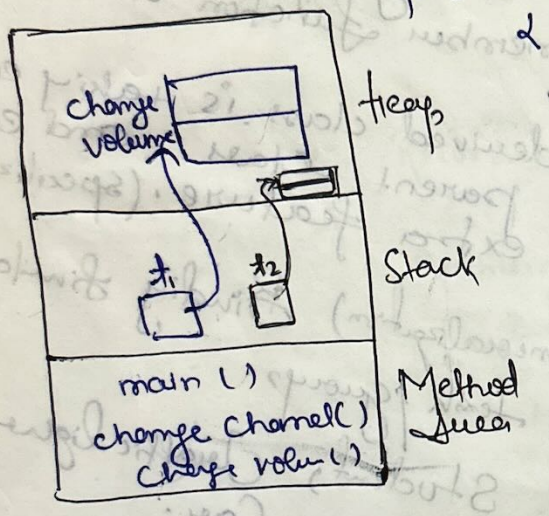
Object :- Anything in the World is an object
↳ Defined in term's of property & behavior



→ A single class have many obj.

```
class Television
{
    private int channel; // property
    private int Volume;
    public void changechannel();
    {
    }
    public void changeVolume() // method behaviour
    {
    }
}
class Test
{
    ps v main()
    {
        Television t = new Television();
                       (obj is created in heap)
        t.changechannel(10);
    }
}
```

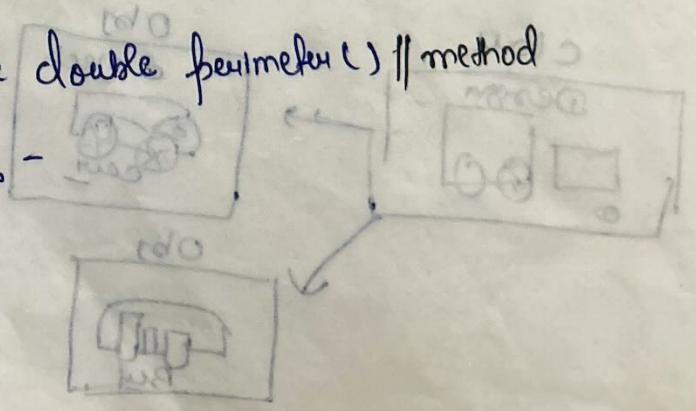
obj ref in stack



How To Write Class

```
class Circle
{
    public double Radius // property
    public double area() // method
    {
    }
    public double perimeter() // method
    {
    }
}
```

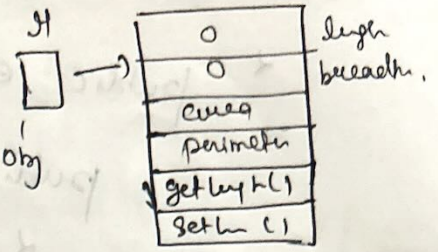
for obj → circle



```
class Circle
{
    public double radius;
    public double area()
    {
        return Math.PI * radius * radius;
    }
    public double perimeter()
    {
        return 2 * Math.PI * Radius;
    }
    public double Circumference()
    {
        return perimeter();
    }
}
class circle1
{
    ps v m (arg[] str)
    {
        Circle c1 = new circle();
        c1.radius = 7;
        System.out.println(c1.area());
        sop(c1.perimeter());
        sop(c1.Circumference());
    }
    {
        Circle c2 = new circle();
        c2.radius = 14;
        sop(c2.area());
        sop(c2.perimeter());
        sop(c2.circumference());
    }
}
```

Every class make seperate class.java file

Data Hiding




```
class Rectangle
{
    private int length
    private int breadth
```

|| other outside class can't access only Indice the class will use

Read [int getLength()
{ return length;
}

write [void setLength(int l)
{ length = l;
}

```
public int area()
{
    return length * breadth;
}
public int perimeter()
{
    return 2 * (length + breadth);
}
}
```

// void setLength(int l)
{ if (l > 0)
length = l;
else
length = 0;
}

```
class Test
{
    Rectangle r = new Rectangle();
```

~~r.length = 10;~~ ~~r.breadth = 20;~~ we can't do it.

So, To access the data we will use function to read and write.

```
r.setLength(10);
r.setBreadth(5);
Sop(r.area());
```

Constructor

Method of a class which is automatically called whenever object is created

```
class Rectangle
{
    private int length;
    private int breadth;

    public rectangle()
    {
        length = 1;
        breadth = 1;
    }
    public Rectangle(int l, int b)
    {
        length = l;
        breadth = b;
    }
}
class Test
{
    psvm(--)
    {
        rectangle r = new rectangle();
        "
        (10,2);
    }
}
```

Java has Inbuild some Constructor i.e default provided by Java Compiler if you define your own then default got removed

Product

	data type
itemno - A27-A7	→ String
name -	String
price -	double
Qty -	Short

Property method

String getItemno ( )

String getname ( )

:

✗ SetItemno (stri item)
↳ we can't change

~~SetItemno ✗~~ ~~Setname ✗~~

✓ set price ( )

✓ set Qty ( )

Constructor

↳ There is no non-parameterized constructor for this problem

product (String itemno) ✓ complexity

✓ product (String itemno, String name)

product (~~all~~)

Subject

Datatype
SubID - String
name - String
Maxmarks - Int
Marksobtain - Int

Student
- rollno
- name
- dept
- Subject

Property Method

Set have get method ( )

set method (SubId)(name) ~~X~~ must

How To Create Array of Obj

constructor & :

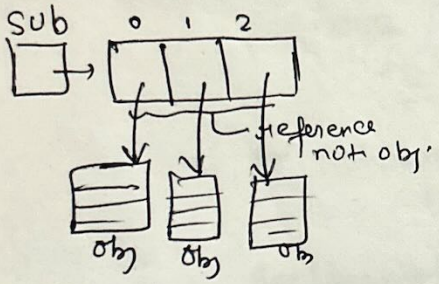
Subject (sid, name)

Subject (- call)

→

```
class Subject
{
    private String SubId ;
    private String name ;
    private int Maxmarks;
    private int marksObtain;

    public Subject ( String SubId, String name, int Maxmarks)
    {
        this.SubId = SubId;
        this.name = name;
        this.maxmarks = maxmarks
        this.marksobt
    }

    public string getSubId ( ) { return SubID; }
    public String getName ( ) { return name; }
    public Int getmaxMarks ( ) { return maxMarks; }
    public Int getMarksobtained ( ) { return marksobtain }

    public void setMaxmarks (int mm)
    {
        maxMarks = mm;
    }
    public void setMarksobtain (int m)
    {
        marksObtain = m;
    }
```

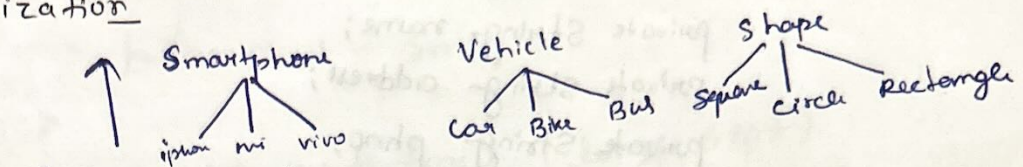
```
boolean isQualified()
{
    return marksobtain >= maxMarks/10*4;
}
public String toString()
{
    return "\n Subject ID :" + SubId +
           "\n Name :" + name +
           "\n Marks obtained :" + Marksobtained;
}
}
public class Scoop3
{
    public static void main (String[] args)
    {
        Subject Sub[] = new Sbject[3];
        Subs[0] = new Subject ("S101", "DS",
                               100);
        Sub[1] = new Subject ("S102",
                 "Algorith", 100);
        Sub[2] = new subject ("S103",
                 "Operating", 100);
        for (Subject S : Sub)
        {
            SOP (S);
        }
    }
}
```



Inheritance

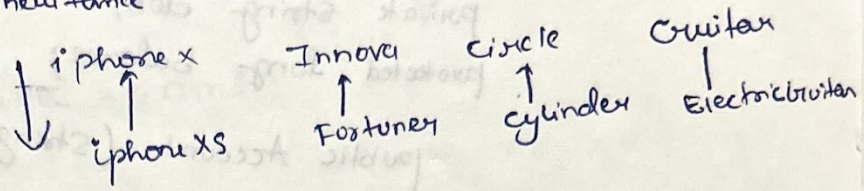
achieve — Interfaces

Generalization



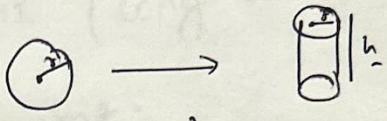
achieve — Inheritance

Specialization →

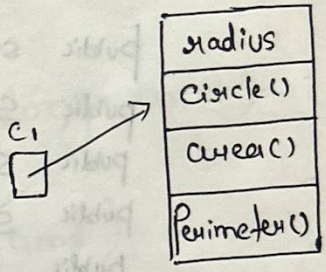


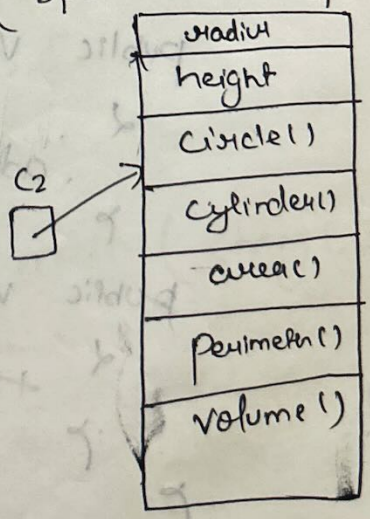
Inheritance is a process of acquiring a feature of an existing class into a new class.

Ex: - ~~this~~ circle & cylinder




```
class Circle    (parent class | supper class)
{
    private double radius;
    public Circle()
    { radius = 0.0;
    }
    public double area() {
    public double perimeter() {
}
class cylinder extends Circle   (Specialized class | Sub class)
{
    private double height;
    public cylinder()
    { height = 0.0;
    }
    public double volume() {
}
class Test
{
    psvm(--)
    {
        Circle c1 = new circle();
        Cylinder c2 = new Cylinder();
        c1.area(); c2.volume();
        c2.area()  c2.perimter();
    }
}
```





Ex:-

```
Class Account
{
    private String accNo;
    private String name;
    private String address;
    private String phno;
    private String dob;
    protected long balance;

    public Account (String acc, String n, String add,
                    String phno, String dob)
    {
        accNo = acc
        name = n
        address = add
        phno = phno
        dob = dob
    }  balance
public String getAccno () { return Accno; }
public String getName () { return name; }
public String getAddress () { return address; }
public String getphno () { return phno; }
public String getDOB () { return dob; }
public long getBalance () { return balance; }

public void setAddress (String Add)
    {
        address = add;
    }
public void setphno (String phno)
    {
        this.phno = phno;
    }
}
```

```
class SavingAccount extends Account
{
    public void deposit (long amt)
    {
        balance += amt;
    }
    public void withdraw (long amt)
    {
        balance -= amt;
    }
}
class LoanAccount extends Account
{
    public void payEMI (long amt)
    {
        balance -= amt;
    }
    public void replay (long amt)
    {
        if (balance == amt)
            balance = 0;
    }
}
public class SCInherit
{
    public static void main (String[] args)
    {
    }
}
```

How to call parameterized Constructor

```
class parent
{
    parent()
    {
        Sop("Non-param of parent");
    }
    parent(int x)
    {
        Sop("param of parent"+x);
    }
}

class child extends parent
{
    child()
    {
        Sop("Non-param of child");
    }
    child(int y)
    {
        Sop("param of child");
    }
    child(int x, int y)
    {
        super(x);
        Sop("2 param of child"+y);
    }
}

public class SuperConst
{
    public static void main(String[] args)
    {
        child c = new child(20);
    }
}
```

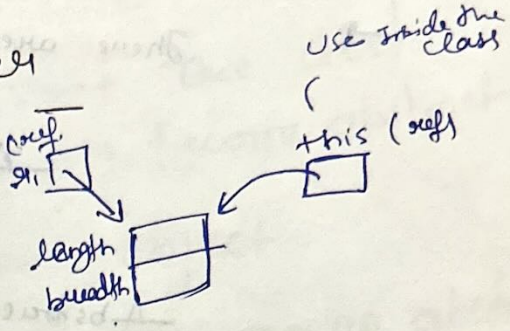
non-param of parent
param of child

child c = new child (10, 20);

↓

1 param of parent 10;
2 param of child 20;

This and Super




```
class rectangle
{
    int length;
    int breadth;

    Rectangle (int l, int b)
    {
        this.length = l
        this.breadth = b
    }
    void display ()
    {
        sop (this.length);
        sop (this.breadth);
    }
}
```

Rectangle r1 = new Rectangle(10,5);

Super

```
class Rectangle
{
    int length;
    int breadth;
    int x = 10;
    Rectangle (int length, int breadth)
    {
        this.length = length;
        this.Breadth = Breadth;
    }
}
```

```
class cuboid extends Rectangle
{
    int height;
    int x = 20;
    cuboid (int l, int b,
            int h)
    {
        super (l, b);
        height = h;
    }
    void display ()
    {
        sop(super.x);
        sop (x);
    }
}
```

~~Method Overloading~~

Abstract Class

→ There are two type of classes

↙ Abstract class Concrete class

Abstract is written before any class then it is abstract class else It is concrete

- ⊙ you can create the obj of concrete classes
- ⊙ you can not create the obj of abstract class but Reference of abstract class is allowed
- ⊙ Abstract class cannot be final/static

```
abstract class super
{
    super(){
        sop("super");
    }
    void meth()
    {
        sop("super Math");
    }
    abstract void meth2();
}
class sub extends super
{
    void meth2(){
        sop("Submeth2");
    }
}
```

```
class Test
{
    psvm(-)
    {
        super s1;
        S1 = new super();   ✗
        Sub s2 = new Sub();   ✓
        (obj of concrete class)
    }
}
```

but Here it is Overrided that abstract method that why we make object

abstract Method

→ method which is not having body there is no flower Blacked or that is undefined is called abstract method.

Abstract class

If a class is having atleast one abstract method then defenetly class become abstract and it doesn't have object.

→ A class can be abstract with no abstract method also

Why we need abstract class

If any class is Inheriting from abstract class then that class also become abstract

To become concreate class it must Override that abstract method of an abstract class.

Ex:
```
abstract class super
{
    public super() { sop("super constructor"); }
    public void meth1() { sop("Meth1 of super"); }
    abstract public void meth2();
}
class sub extends super { @override
    public void meth2() { sop("sub meth2"); }
public class Abstractex {
    psvm(-)
    {
        Super s = new sub();
        s.meth1();
        s.meth2();
    }
}
```

Super constructor
Meth1 of super
Sub Meth2

Abstract class

abstract class KFC ~~{ }~~ → abstract class

```
{
    KFC ()
    {
    }
    void makeItem ()
    {
    }
    abstract void billing ();   — It is done by who is the franchise
    abstract void offer ();     — It my choise
}
```

It my choise who to do what software we have to ask KFC don't give any rule for that

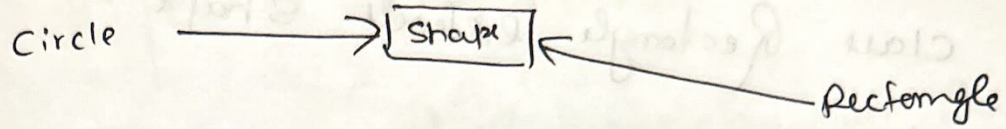
Concrete class

```
class MyKFC extends KFC
{
    MyKFC ()
    {
    }
    void billing ()  — overrided
    {
    }
    void offer ()  — overrided
    {
    }
    void festiv offer ()
    {
    }
}
```

KFC K = new MyKFC();

ref. obj.

K.makeitem();
K.billing();
K.offer();
K.festiv offer();



Shape have
- ⊙ No properties.
- ⊙ perimeter — Method
  - and have to be abstract bcz we can't find abstract conic perimeter.
- ⊙ area()

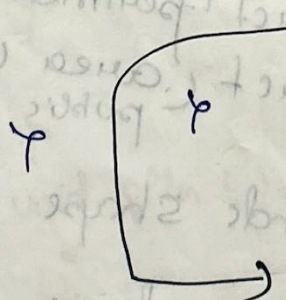
Circle extends shape
- It must override perimeter and area to become concrete class.
  - property → radius
  - Method → perimeter, area

Rectangle extends Shape
- It must over ride perimeter and area
- property → length, breadth

→ abstract class Shape {
    public double
    abstract perimeter();
    abstract area();
    public double
}

class Circle extends Shape {
    double radius;
    public double perimeter()
    { return 2 * Math.PI * radius;
    }
    public area(){
    return Math.PI * radius * radius;
    }
}

```
class Rectangle extends Shape
{
    double length;
    double breadth

    public double perimeter () {
        return 2 * (length + breadth);
    }
    public double area () {
        return length * breadth;
    }
}
public class Test
{
    psvm (-)
    {
        Rectangle r = new rectangle();
        r.length = 10;
        r.breadth = 5;
        Sop (r.area());
    }
}
```



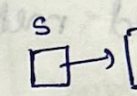
or

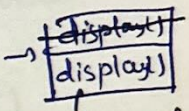
```
Shape s = r;
Sop (s.area());
```

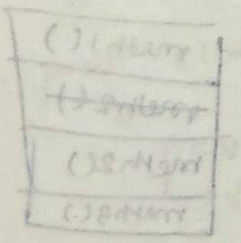
Method Overriding

redifining a method of super class in subclass
Method overriding is acheived in Inheritance

```
class super
{
    public void display()
    {
        sop("Hellow");
    }
}
class sub extends super
{
    public void display()
    {
        sop("Hellow world");
    }
}
class Test
{
    psvm main()
    {
        Super S = new Super();
        S.display; —— Hellow
        Sub Sb = new Sub();
        Sb.display() → Hellow world
                        (, overrided Hellow
    }
}
```







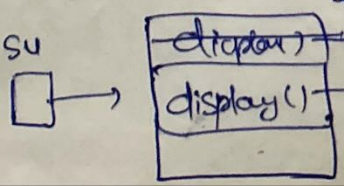
It is a concept of Java in which we can create ~~multiple methods~~ a subclass method same as parent class method.

Dynamic Method dispatch

```
class Test
{
    psvm(-)
    {
        Super SU = new Sub();
    }
}
```

ref of super class

object of sub class



Hellow world

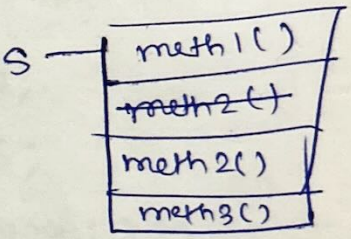
SU.display()

the method will called depend upon obj

A super class reference holding the object of sub class and calling overrided method (i.e. method of object not of reference) is called Dynamic Method dispatch.

Dynamic Method Dispatch useful for acheiving Run time polymorphism using Method Overriding.

```
class super
{
    void meth1() {}
    void meth2() {}
}
class sub extends super
{
    void meth2() {}
    void meth3() {}
}
class Test
{
    psvm(String[] args)
    {
        super S = new sub();   ✓
        // sub S = new super();   ✗
        S.meth1();   ✓
        S.meth2();   ✓
        S.meth3();   ✗ → It is in the class but we can't access because the reference is super class
    }
}
```



PolyMorphism

↳ One name different action

poly → Many morphism → form

↳ It is acheived by overloading and overriding

```
package Overloading
class MAX
{
  public int MAX (int a, int b)
  {
    return a>b ? a:b?
  }
  public int MAX (int a, int b, int c)
  {
    if (a>b && a>c)
      return a;
    else if (b>c)
      return b;
    else
      return c;
  }
}
public class Overloading
{
  psvm(-)
  {
    Demt t = new Demt();
    t.max (10,5);
    t.max (10,15,5);
  }
}
```

name is same by having different action (polymorphism)

↳ It is compile time polymorphism bcz at compile time it will identify which method will called

Overloading will acheive in a single class.

```
Package Override;
Class super
{
  protected void display()
  {
    sop ("=super display");
  }
}
class sub extends super
{
  public void display()
  {
    sop ("sub display");
  }
}
public class Override
{
  psvm(-)
  {
    Super s = new super();
    s.display();
    Super s = new sub()
    s.display();
  }
}
```

super.display

Sub display (DMD)

name is same but have different Action (polymorphism) but it is done at run time.

Overriding will acheive in two classes or more than one class

Inner class

we can define a class Inside a class is called nested class / Inner class

Need

If you have a ~~that is~~ class that is very big very complex for making it simple then we can write in a form of class itself i.e. Inner class.

Types:-

i) Nested Inner class
ii) Local Inner class
iii) Anonymous inner class
iv) Static inner class

① - Nested Inner class

```
class Outer
{
    int x = 10;  ⟶ Just like a global variable for Inner class
    class Inner
    {
        int y = 20;
        void innerDisplay()
        {
            Sop(x);  ✓
            // It can acces the value of outer class
            Sop(y);
        }
    }
    Void outerDisplay()
    {
        Inner i = new Inner();  Inside the outer class we can create the obj of Inner class.
        i.innerDisplay();
        Sop(i.y);  — i is obj of inner class can it access the y
    }
```

Directly outerclass cannot acces the the variable/members of Inner class
we can acess it by making obj.

conclusion

Inner class can acess the data member of Outclass Directly but

Outclass can acess the datamember of Innerclass by creating object

```
class Test
{
    psvm(-)
    {
        Outer O = new Outer();
        O.outerDisplay();
        Outer.Inner i = new Outer().new Inner();
    }
}
```

outside the class we can acess the Inner class. (but it is useless)

→ compiler make class - Outer.class

but every class will create a class.file

But for inner class

↳ Outer $ Inner.class

Local Inner class

```
class Outer
{
    class Inner
    {
        void innerDisplay()
        {
            Sop("Hellow");
        }
    }
    Inner.i = new Inner()
    i.innerDisplay();
}
```

If you want any class to inherit from some existing class or Implement an Interface and that class is useful only for this method not for outside then it is Local inner class

Anonymous Inner class

It is defined at the time of creation of object itself.

It is useful for abstract or Interface class

```
abstract class My
{
    abstract void display();
}
class outer
{
    public void meth()
    {
        My m = new My()
        {
            public void display()
            {
                Sop("Hellow");
            }
        };
        m.display();
    }
}
```

Static Inner class

→ Static inner class are the static member of outer class and the object of Static inner class are created outside the outerclass It it can be acess from anywhere and we don't have to create the outerclass obj.

```
class Outer
{
    static int x = 10;
    int y = 20;
    static class Inner
    {
        void display()
        {
            sop(x); ✓
            sop(y); ✗ → It cannot access non-static
        }
    }
}
class Test
{
    psvm(-)
    {
        Outer.Inner i = new Outer.Inner();
        i.display();
    }
}
```

Static

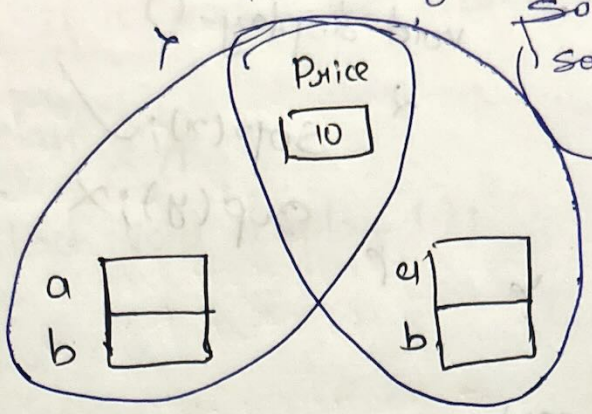
It is used for representing Metadata (i.e. data about data)

Static member are useful for representing information or data related to a class

⊙ static member is shared to all the obj of class

```
class HondaCity
{
    static long price = 10;
    int a, b;
    static double OnRoadPrice(String city)
    {
        Switch(city)
        {
            case "delhi":
                return price + price * 0.1;
            case "mumbai":
                return price + price * 0.09;
        }
    }
}
```

```
class Test
{
    psvm main ()
    {
        Hondacity h1 = new Hondacity();
        Hondacity h2 = new Hondacity();
        sop(h1.price); → 10
        sop(h2.price); → 10
        If h1.price = 20;
        both will print 20
    }
}
```




```
sop(Hondacity.price);
    ↳ 10
```

Static member can be accessable by just using class name

Static member are created under method area

Static method

↳ Static method will give the result after compution that value

① Static method belong to a class they are common for all the object they can be called by just using class name

Static method can access only static member not any non-static

```
sop(Hondacity.onRoadPrice("Delhi"));
```

Interface

In Abstract class used to acheive ① polymorphism ② Inherit

Interface is completely used for acheiving polymorphism

It doesn't have anything to give to sub class

So, when we had to achive only polymorphism doesn't have to give or borrow anything So we used Interface.

If an abstract class doesn't have anything to share with sub class It forces the subclass to override all it's method so, it purpose is only to acheive polymorphism.

For such an abstract class we write Interface

Ex :-
```
abstract class Test1
{
    abstract public void meth1();
    abstract public void meth2();
}
class Test2 extends Test1
{
    public void meth1()
    {
    }
    public void meth2()
    {
    }
}
Test1 t = new Test2();
```
ref superclass, obj of subclass

⟹ Interface

by default abstract + public
```
Interface Test1
{
    void meth1();
    void meth2();
}
class Test2 implement Test1
{
    public void meth1()
    {
    }
    public void meth2()
    {
    }
}
```

① when we say Extend a class can extend from Only one class but

② In Implementing a Interface it can implement more than one (Multiple Interfaces)

Ex:-

```
package Interfacepractice;

Interface Test
{
    void meth1();
    void meth2();
}
class My implement Test
{
    public void meth1()  — overrided
    {
        sop("Meth1 of class My");
    }
    public void meth2()  — overrided
    {
        sop("Meth2 of class My");
    }
    public void meth3()
    {
        sop("Meth3 of class My");
    }
}
public class Interfacepractice
{
    public static void main(String args[])
    {
        Test t;  — ref of super class
        Test t = new My()
                 Obj of sub Interface
        t.meth1();
        t.meth2();
        t.meth3()
```

Acheived

DMD
run time polymorphism

bcz ref is of super

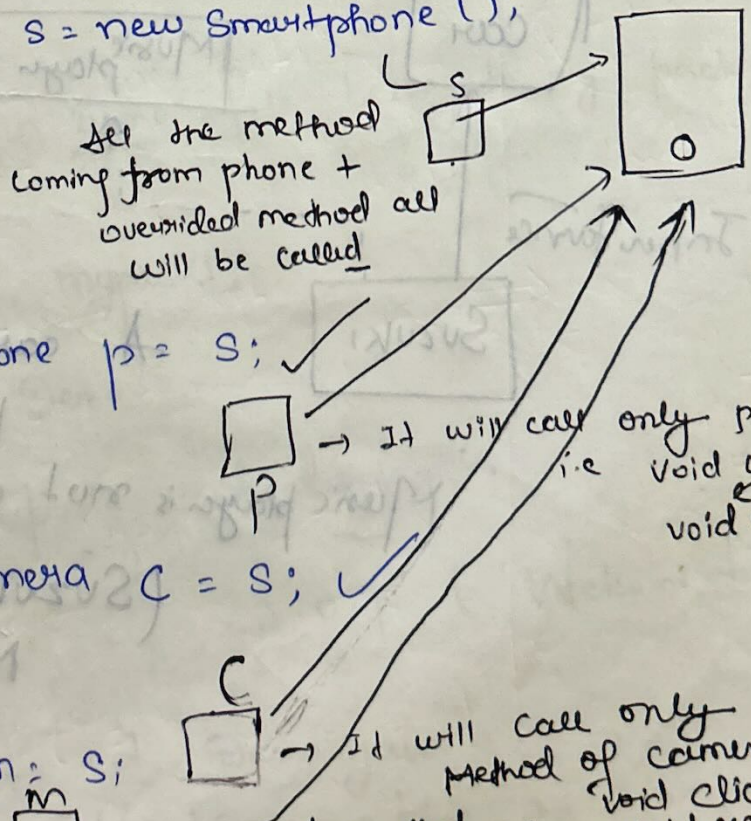
Interface

```
class phone
{
    void call()
    {
    }
    void sms()
    {
    }
}

Interface camera
{
    void click();
    void record();
}

Interface MusicPlayer
{
    void play();
    void pause();
    void stop();
}
```

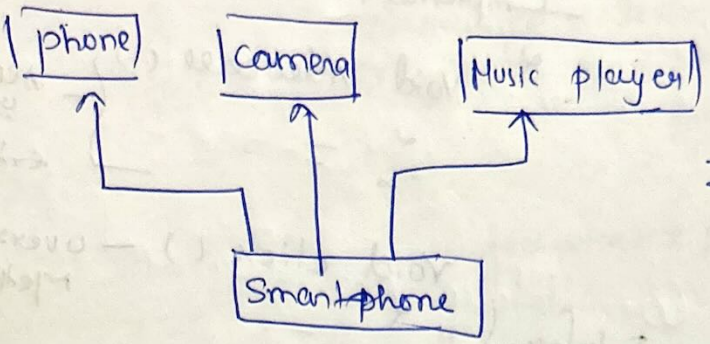
```
class smartphone Extends Phone
Implements camera, MusicPlayer
{
    void videocall()
    {
    }
    void click()
    {
    }
    void record()
    {
    }
    void play()
    {
    }
    void pause()
    {
    }
    void stop()
    {
    }
}
```

- videocall() — Set feature of phone + extra feature
- click() — overrided Method
- Interface class
- record() — overrided method
- play(), pause(), stop() — overrided methods

→ `Smartphone S = new Smartphone();`

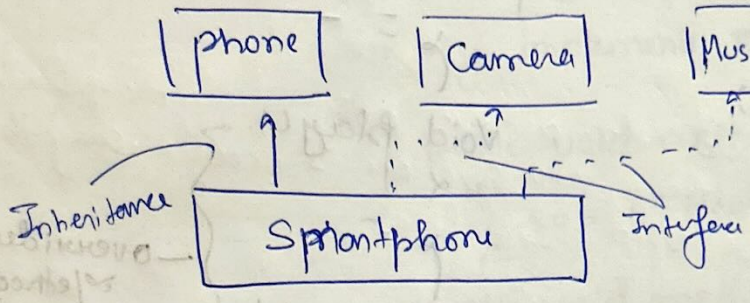


Multiple Inheritance — Interface



— In c++ only

It is all phone, camera, Music

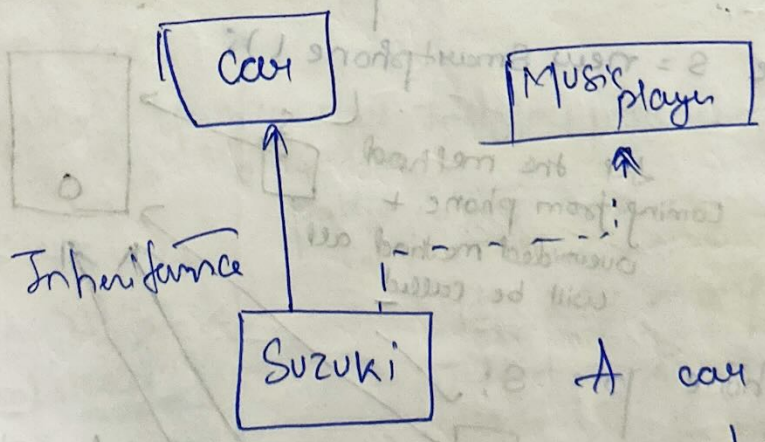


— In java

It can act's as Camera, Music player

So, these one Interface

Not multiple Inheritance



A car is a Suzuki but a

Music player is not a Suzuki

(Suzuki implement Music player)

Packages

package is a collection of classes, Interfaces or other packages

used for Organising java project.

```
// import java.lang.String;
import java.lang.Runnable;
class Test
{
    psvm(String args[])
    {
        java.lang.String str = new java.lang.String("Hello");
    }
}
```

(we like this) ——

How to Create Our Own packages

This Demo file belong to my package mypack1;

```
Package mypack1;
class Demo
{
    public void display()
    {
        S.O.P("Welcome to Demo");
    }
}
```

javac -d . Demo.java

path for package: java -d c:/ Demo.java → This create package in current Directory.

Adding more class to same package-

→ package mypack1;

class Demo

{

public void display()

{

sop("welcome to Demo");

}

}

→ Save as Demo2.java

Again package mypack1;

class Demo2

→ Save

cd mypack1

\MyJava\mypack1> Demo.class

Javac Demo2.java

Now I want to Include this Demo2 to my package

\MyJava > del *.class

Javac -d. Demo2.java

cd mypack1

\Myjava\Mypack1> Demo.class
Demo2.class

Subpackage

```
package mypack1.inner;
public class Demo3
{
    public void display()
    {
        Sop("welcome to
            Demo3");
    }
}
```

Save Demo3.java

cd mypack1

\Myjava\mypack> Demo.class
Demo2.class

cd..

\Myjava> javac -d. demo3.java

cd mypack1

\Myjava\Mypack1> Demo.class
Demo2.class
Inner

cd Inner

\Myjava\Mypack1\inner> Demo3.class

Use those package made by me

```
import mypack1.Demo;
import mypack1.Demo2;
import mypack1.inner.Demo3;
class Test
{
    p s v m (String args[])
    {
        Demo d1 = new Demo();
        d1.display();
        Demo2 d2 = new Demo2();
        d2.display();
    }  Demo3.d3 = new demo3();
       d3.display();
```

O/P → Welcome to Demo
Welcome to Demo2

we are able to do this bez package are also at the same folder

(If it is in diffent then we had to set the class path)

Access Modifiers

Use for specifying the access level or visibility of the member of the class

Types :-
- Default
- private
- protected
- public

class Demo { — only → Default, public

{
- data member → int x;
- method → void show()
- class → class Inner { }

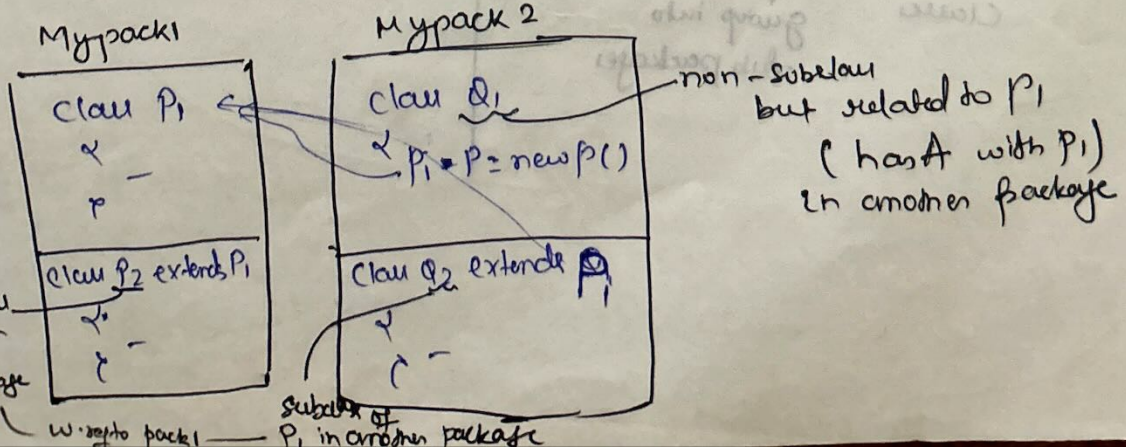
— Anything → private, Default, protected, public

class Demo1 (How to use these class)

{

(non-subclass) hasA → class Test { Demo1 d = new Demo1(); } (by making object)

isA (subclass) → class Demo2 extends Demo1 { } (By Inheritiy)



Access Modifiers

	default	Private	Protected	Public
Same class	✓	✓	✓	✓
Same Pack Sub class	✓	✗	✓	✓
Same pack non-Sub class	✓	✗	✓	✓
Diff pack Sub class	✗	✗	✓	✓
Diff pack non-Sub class	✗	✗	✗	✓

Package Naming Convension

URL: http:// www.univ.com

Com.univ.academic

University

- Student
- Faculty
- Course
- Book
- Library

↓ Classes

under main package

- academic
- admission
- Examination

↓ group into sub packages

Bank

http:// www.citi.com

↳ Com.citi.account

Com.citi.Loans

Exception Handling

↳ Run time Errors

Type of Errors

Programmer
- ·) Syntax Error → This type of error remove by compiler
- ·) Logical Error → remove by debugger
  - ⊙ If there is no typing mistake & if a programer is compiling and tried to run and test the program & the program is not giving proper Result.

```
int x, y;
① a = 10 ⓪ no;
② Z = x + y;
   └ not initialize
```

⊙ x = -b/2a } x = -b/(2*a);
└ not given

```
⊙ for(int i=①; i<
  A.Length; i++)
  {
   Sop(A[i]);
  }
```

User
- ⊙ Run-time Errors (Exception)
  - ↳ Exception Handling
  - ↳ By Mishandling of the program

by :- ·) Entering Invalid Input
·) Internet ·) File.

Soln is - In case of these error a message had to given to the user that you had done these mistake.

⊙ (Inform the User)

Ex:-

```
class Test
{
    psvm (String args[])
    {
        int a,b,c;
        try{
            a = 10;
            b = 0;
            X c = a/b ; — Error  10/0 = ∞
            — X Sop ("Result is" + c);
        }
        Catch (Arithematic Exception e)
        {
            Sop ("Division by zero");
        }
    }
}
```

build class in java

It will not execute

In try block if there is any error then it go to catch block otherwise it will be in try block and execute whole

Ex:-

```
try
{ int A[] = {10, 0, 2, 3, 5};
  int r;
  r = A[0]/A[1];   — 10/0 ; (Error)
  Sop (r);
  Sop (A[10]); X (Error)
}
Catch (Arithmetic Exception e)
{ Sop(e);
}
Catch (ArrayIndexOutOfBoundary Exception e)
{ Sop (e);
}
```

This should be subclass
↳ If it is super then it hides this catch

↳ This should be super class

Nested try & catch

```
try
{
    int A[] = {10, 0, 8, 5, 3};

    try{
        int x = A[0]/A[1];
        sop(x);
    }
    catch (ArithmeticException e)
    {
        sop(e)
    }

    sop(A[10]);
    }
    catch (ArrayIndexOutOfBound e)
    {
        sop(e);
    }
}
```

error

error

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

finally — It directly execute allthough there is any error or not

for clean-up source

Exception Clause

* Every user defin exception must Inherit from
① Exception class

```
psvm()
{
    try
    {
        sop(10/0);
    }
    catch (Exception e)
    {
        sop(e);
    }
    finally
    {
        sop("DONE");
    }
```

If want / If not

Object (Mother class of all Java class)

Throwable class

Exception class — Error class

Class Exception

* String getmessage()
② String tostring()
③ void printstack Trace()

```
sop(e.getmessage())
sop(e);
e.printstacktrace();
```

① checked Exception
└ Must handle by try & catch

② unchecked
└ If you want to handel this then you can it is not mandetory.

- ClassNotFoundExcept
- IOException
- InterruptedExcept
- NumberFormat Excepti
- RuntimeExceptio
  - Arithmetic Exc
  - IndexOutOfBound Exc
  - NullPointer Exc

```
try{
    =
    =
}
catch (ArithmeticException e)
{
    —
}
catch (Exception)
{
    =
}
```

Sub

Super

It can handel any exception

Throw (vs) Throws

Throwing an Exception

It say this method throws exception

```
int area (int l, int b) throws Exception
{
    if (l<0 || b<0)
        Throw new Exception("-ve");
    int a = l*b;
    return a;
}
```

Checked must try & catch

If you are not handling in the same line (bcz it doesn't make sense that throw & handle in same line)

```
int meth1 () throws Exception
{
    try{
        int a = area (-10, 5)
        Sop (a);
    }
    Catch (Exception e)
    {
        Sop (e);
    }
}
```

If you are not catching here

```
psvm ()
{
    try
    meth1 ();
    Catch
}
```

So, the method which throw checked Exception they must have in the signature that they may throw so and so Exception.

And whoever is calling it must handle it.

And If you don't want watnt to Catch to the calling function then, write throws there also

How to define Our One class for Throw Exception.

class NegativeDimensionException extends Exception
(must bcz it's our own Except class)

{
public void toString()
{
return "Dimension can not be negative";
}
}

int area (int l, int b) throws negativDimenExcep
{
if (l<0 || b<0)
throw new NegativeDimensionException();
int a = l * b;
Return a;
}

Static void meth1() throw NegativDimensionExcep
{
sop ("area is" + Area(-10, 5));
}

public static void main (String[] args)
{
try
meth1();
catch (NegativDimensionException e)
{
sop (e);
}
}

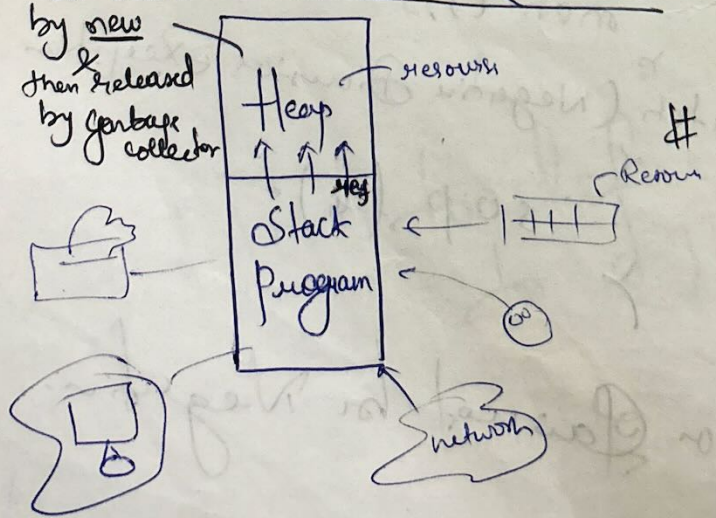
→ Dimension cannot be Negativ.

Finally

```
public class finallydemo
{
    static void methi() throws Exception
    {
        try {
            throw new Exception
            {
            }
        finally
            {
                sop ("Final message");
            }
        }
    psvm ( ) throws Exception
        {
            methi();
```

Try with Resource

Q What is Resource



Whenever if require it acuire it and then Return if don't want & if you don't Return program link you are still using it

Why we Need Finally

```
int meth1() throws Exception
{
    FileReader f;
    try {                                   — resource
        f = new FileReader("my.text");      — open file
        // use file
        f.close();
        return result;
    }
    finally { f.close(); }
}
```

In case there is using Exception file here the it doesn't execute these line

f.close() must to execute

then, If f.close() is not executed then file is not close it not release then it still holding that file without releasing it & clean up not done.

∴

f.close in in finally

Try with ~~finallyResource~~

```
try meth1() throw Exception
{
    try (FileReader f = new FileReader("myText");   — This resource open | create with try it self
    {
        // use file
        return result;
    }
}
```

It automatically Close once method is ends

You need finally block for cleanup