OOP

Classes → group of entities like object.

↓

* attributes (properties)

+

* functions (behaviour)

for ex

pen (class) → colour → attribute / property

→ change colour → function

* so basically class will give us a blue print which state which properties & function is present

ex

```
public class OOP { }
class pen {
    string color;
    int tip;
void setcolor (string newcolor)
{ color = newcolor; }
void settip (int newtip)
{ tip = newtip }
}
```

* in public class OOP {

```
public static ... {
pen p1 = new pen();
p1.setcolor ("Blue");
p1.settip (5);
```

S.o.p (p1.setcolor);

* Access modifiers

Access modifier	within class	within package	outside package by subclass only.	outside package
private.	Y	N	N	N
Default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

Ex

```
class BankAccount {
    public String username;
    private String password;
    public void setname (String newname)
      { username = newname; }
    public void setpwd (String pwd)
      { password = pwd; };
}

P.S.V.M()
{
    BankAccount myAcc = new BankAccount();
    myAcc.username = "Atharva";
    myAcc.password = "A1B2"; ✗ (Not possible due to private)
    myAcc.setpwd ("A1B2"); → possible
```

Getter & Setters

Get :- to return the value
Set :- to modify the value

- If some properties are private then it is not accessible to anyone
→ then to access these private properties we define a fun known as getter

- This :- this keyword is used to refer to the current object
→ जीरत bhi object ke lye getcolour() call hoga thise usi object को replace कर देगा

it is mostly use when we have variables having same name.

→
```
class pen {
    int tip;
    void setTip(int tip)
    { this.tip = tip; }
}
```
replaced by mypen

p.s.v.m --
```
pen mypen = new pen()
```
→ obj

Code

```
p.s.v m - - {
        (
        ;
}
class pen {
 private color;
 private tip;
 string set get color()
   { return this. color; }
  int get tip() {
   return this. tip; }
 void set color (String color) {
 this.color = color; }
 void set tip (int tip) {
    this. tip = tip; },
}
p.s.vm ( ) -.
 Pen p1 = new pen ();
 p1. set color ("Blue");
  S.O.P (p1. get color());
 p1. Set tip (5);
 S.O.P ( p1.get tip());
```

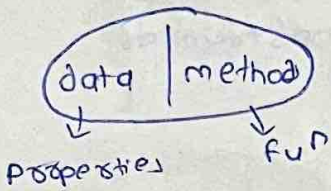
S.op (p1.color)

not possible coz color is private, we have to use getters & setters method for this

* Encapsulation

→ Encapsulation is defined as the wrapping up of data & method under a single unito It also implements data hiding
↘ using access specifiers

class



* Constructors

→ It is a special method which is invoked automatically at the time of object creation

- constructors have the same name as class or struct
- cnsts dont have retur type (Not even void)
- cnsts are only called once; at object creation
- Memory allocation happens when constructor is called

but in that case we donot initilize any value

Ex

```
class student {
    string name;
    int roll;
    student (string name) {
        this.name = name;
    }
}
public class oop {
    p.s.v.m(...) { student s1 = new student("Atharva");
```

← If we don not creatre a constructo
• Java will automatically create one default constructor.

* Types of constructor

1) Non- parameterized
2) Parameterized
3) copy constructor

* Java make default constructor only when it does not have any other constructor

* constructor overloading

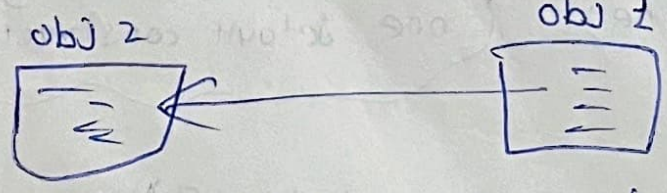
- It is a type of polymorphism.

```
class Student {
    Student() { s.o.p(.....) }
    Student (int roll no) { this.roll no = roll no; }
}
Public class APP {
    P.S.V.M - - - { Student s1 = new student();
                    Student s1 = new student (5); }
```

- we create multiple objects based on it a suitabe constructor will called

* Copy Constructor

When we have 2 object → then copying all the properties of obj 1 in obj 2.



∴ obj 2 is a copy of obj 1

```
class Student {
    String name;
    int roll no;
    String password;

    Student() {
        S.op("constructor");
    }

    Student(String name) {
        this.name = name;
    }

    Student(Student s1) {
        this.name = s1.name;
        this.roll no = s1.roll no;
    }
}

pubic class oop {
    p.s.v.m() {
        Student s1 = new Student() {
        s1.name = "Shradha";

        Student s2 = new Student(s1);
```

---

* Shallow & Deep copy

- Shallow ↓
  - Sirf references copy करते हैं
  - ↓
  - नया object नहीं बनाते
  - ie marks object
  - ↓
  - Changes dont reflect → original value नहीं milega
- Deep copy ↓
  - will create new marks array
  - Changes reflect
  - ↓
  - orginal value milege

Shallow copy

```
Student (Student s1)
{
    marks = new int[3];
    this.name = s1.name;
    this.roll = s1.roll;
    this.marks = s1.marks;
}
```

↑ Shallow copy.

Deep copy

```
Student (Student s1)
{
    marks = new int[3];
    this.name = s1.name;
    this.roll = s1.roll;
    for (int i=0; i<marks.length; i++)
    {
        this.marks[i] = s1.marks[i];
    }
}
```

↑ create new array ie deep copy

* Destructor

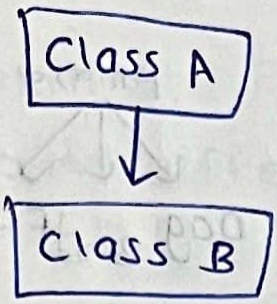
In C++, there is special fun which delete the memory of an object

But in Java, it has garbage collector.

Garbage collector :- jo bhi variable, object, array which is not in use in program so it delete that automatically

So, we do not need to specify destructor in Java, it do it by ourself

Inheritance



← Parent / Base class

← child / derived class.

Parent class की सभी obj aur properties derived class में present रहेगी

⇒ It is when properties & methods of base class are passed on to a derived class

```
Class Animal {
}

class Fish extends Animal {
}
```

← how to define

---

* Single level Inheritance.

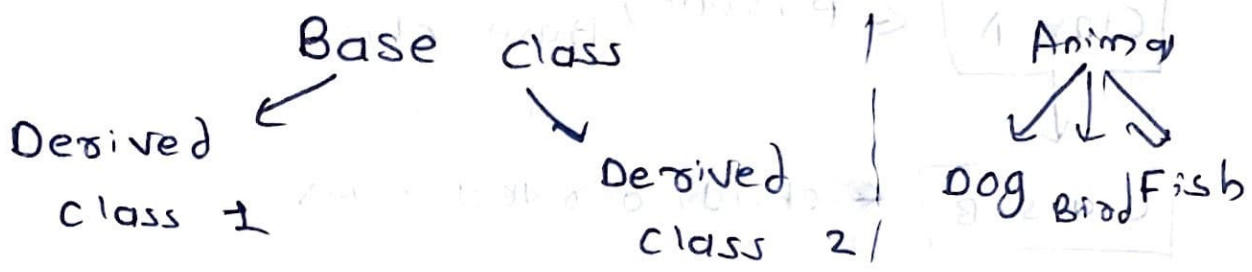
Base class
↓
Derived class

* Multi-level Inheritance

Base class
↓
Derived class
↓
Derived class 2

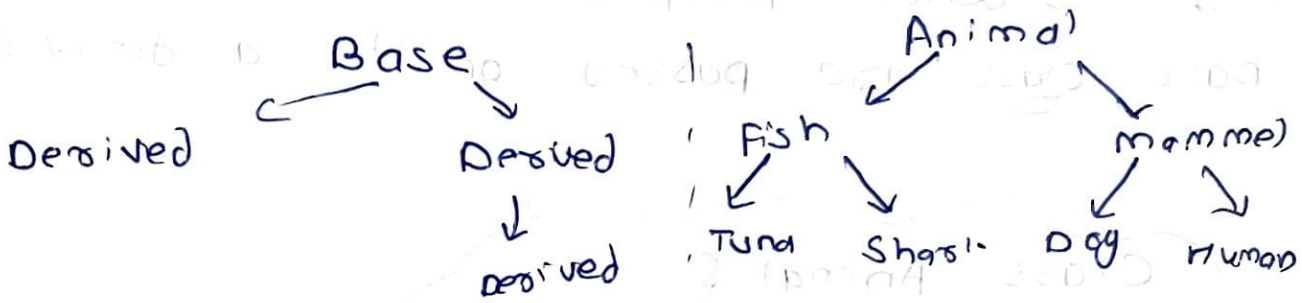
Animal
↓
Mammals
↓
Dog / Cat

* ~~Hierarchical~~ Inheritance

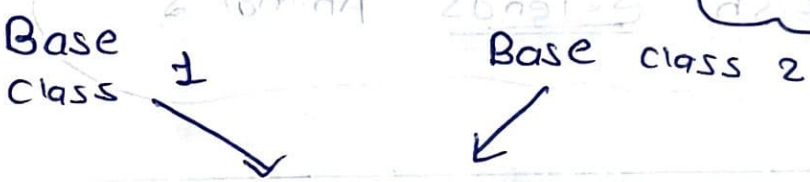


→ single base class have multiple derived class

* Hybrid Inheritance



* Multiple Inheritance → In java it can be implemented using Interface



Polymorphism (Many forms)

→ When we try to do similar thing in multiple forms

- Compile time polymorphism (static)
  - method overloading
- Run time polymorphism (Dynamic)
  - method overriding

- **method overloading**

multiple functions with the same name but different parameters

Ex

```
class Calculator {
    void sum (int a, int b)
    {  -  }
    void sum (float a, float b)
    {  -  }
}
```

```
int sum (int a, int b, int c)
{ -- }
```

- **method overriding.**

Parent & child classes both contain the same function with a different definition

ie

a fun which is both ie parent class & base have & in both class it have same name, same parameters but working is differed

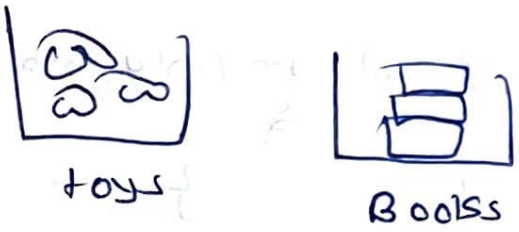
Ex

```
class Animal {
        void eat() { - }
}
class deer extends Animal {
        void eat() { -- }   ← overridding is happen
}
psvm
{ deer d = new deer(); (obj)
  d.eat(); → it will call fun of child class
}
```

coz obj is of child class

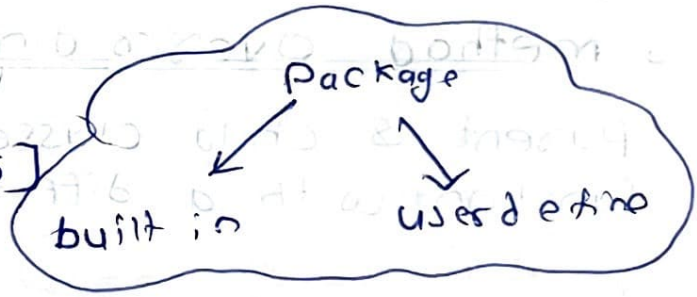
Packages

Package is a group of similar types of classes, interfaces & sub-packages



ex

- Search → • type
  • paste
  • delete
- ~~buy~~ cart → • delete
  • add
- Ratings → • give rating ()
  • rating () [1-5]
  • delete



Package

classes
subpackages
interface

ex import javar util.*;
↓
built in package

Package myPackage;
→ user define

Abstraction

Hiding all the unnecessary details & showing only the important parts to the user

↙ ↘

Abstract classes — Interface

- It gives idea
- Not implementation

Abstract classes

- Cannot create an instance (obj) of abstract class.
- can have abstract / non-abstract methods
- can have constructors
- must contain atleast 1 abstract method

```
abstract class Animal {
    String color;
    void eats() { "eats" }
    abstract void walk();
}

Animal() {
    color = "brown";
}
```

```
class Horse extends Animal {
    void walk()
        { "walks"; };
}
```

← it need to override the abstract method

```
    void changeColor() {
        color = "dark brown";
    }
```

* Here we cannot create a obj of Animal class but can create a obj of Horse or any additional subclass

* Use of constructor:

- if we donot specify the fun to change color it will the brown color to horse

i.e by default brown color dega jb tk Hum change नहीं करेंगे

* जब भी हम child class का obj बनाते हैं तो पहले parent class का constructor call होता है

i.e constructor is used to initilize some varbiles

Interface

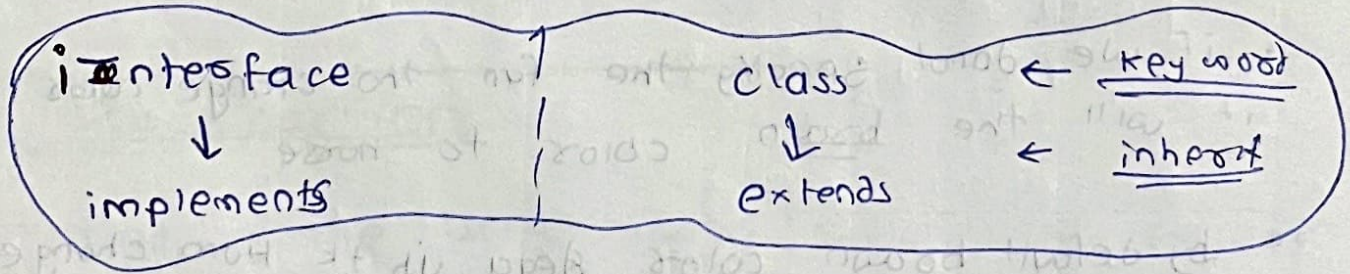
- Class → blueprint of object
- Interface → blueprint of class

M800 ← blueprint ← Blueprint
(class) (Interface)

Car [wheels, speed, engine] -- Interface
↓
Maruti 800 (class)
↓ ↓ ↓
[car1 car2 car3
M800 M800 M800] objects

---

* Inheritance is used for
  - multiple Inheritance
  - To achieve total abstraction
    ⇓
    only abstract method



* Properties

- All methods in interface are public, abstract & without implementation
- used to achieve total abstraction

- variables in the interface are final, public & static

* In Interface it is implemented by class ie all methods & funs are override by class

* For multiple Inheritance

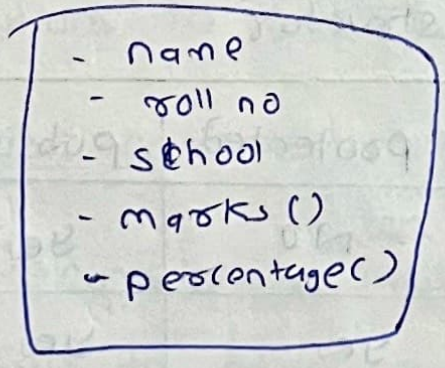
```
interface Herbivore { .. }
interface Carnivore { .. }
class Beer implements Herbivore, Carnivore {
    override all methods & fun
}
```

---

* Static Keyword

Static keyword in Java is used to share the same variable or method of a given class

eg: class Student ⟶ S1, S2 → similar ie school, S3



- so make it static ie which is remained unchanged.

What we can make static

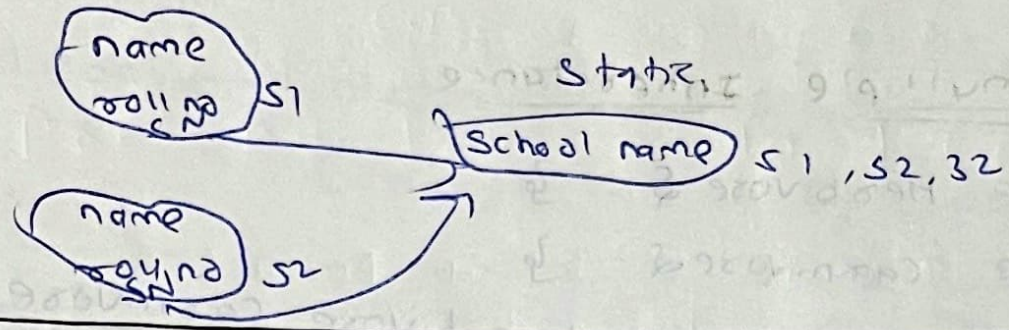
- properties :- variable, string, ---
- function
- blocks { .. }
- Nested class ⟶

```
class A {
    class B { }
    class C { }
}
```

Static obj memory # ↓ bars ही दाखवते

- if we change if for obj 1 then by default it will change for all the objs

Ex



* Super keyword

- this → used to point current object

- Super keyword is used to refer immediate parent class object
  - to access parent's properties
  - to access parent's functions
  - to access parent's constructor

	private	default	protected	public
class	NO	yes	NO	yes
Nested class	yes	yes	yes	yes
constructor	yes	yes	yes	yes
Method	yes	yes	yes	yes
Field	yes	yes	yes	yes