orient electric

DATE / /

OOPS (Love babbar videos)

Video : 1

Class → Nothing but "user-defined data-type".

Object → instance of class.

⇒ If a class is empty, then the size of the object of the class is "1".

⇒ We can access class members using "." operator.

Access modifiers:

→ Public
→ Private
→ Protected.

In default they are "Private".

orient electric

DATE /

→ Getter and Setter functions are useful for getting and setting private data members of the class.

→ Padding and greedy alignment. (Read once)

→ Dynamic allocation.

Hero *b = new Hero;

↳ class name.

→ Constructor :-

↳ does not input parameters

↳ does not have return value.

Parameterised constructor :-

↳ containg i/p parameters.

→ this keyword

↳ stores the address of the current object.

orient electric

DATE / /

→ If you write a parameterised constructor then the default constructor will be gone so you need to write default constructor manually if you write parameterised constructor.

→ Copy Constructor (Inbuilt)

hero s(60, 'A');

hero r(s) → object r → copies object `s`.

⇒ default copy constructor alway does shallow copy and we can do deep copy by custom copy constructor.

Copy Assigment operator

```
hero s (60, 'c', ram)
hero r (70, 'a', sam)

s = r;

then s.health = 70
     s.level = 'a'
     s.name = sam.
```

Destructor :-

```
~Hero () {
    cout << "destructor called" << endl;
}
```

-> for statical allocated -> destructor is automatically called.

-> for dynamical -> destructor should be manually called.

```
hero *b = new hero ();
delete b  -> this is manual destructor.
```

orient electric

DATE / /

Constant keyboard. → Read

initialisation list

↓

not so useful.

Static keyword :-

Static member

↳ creates a data members that belongs to class.

↓

to access this there is no need of object.

Static function:

No need to create object.

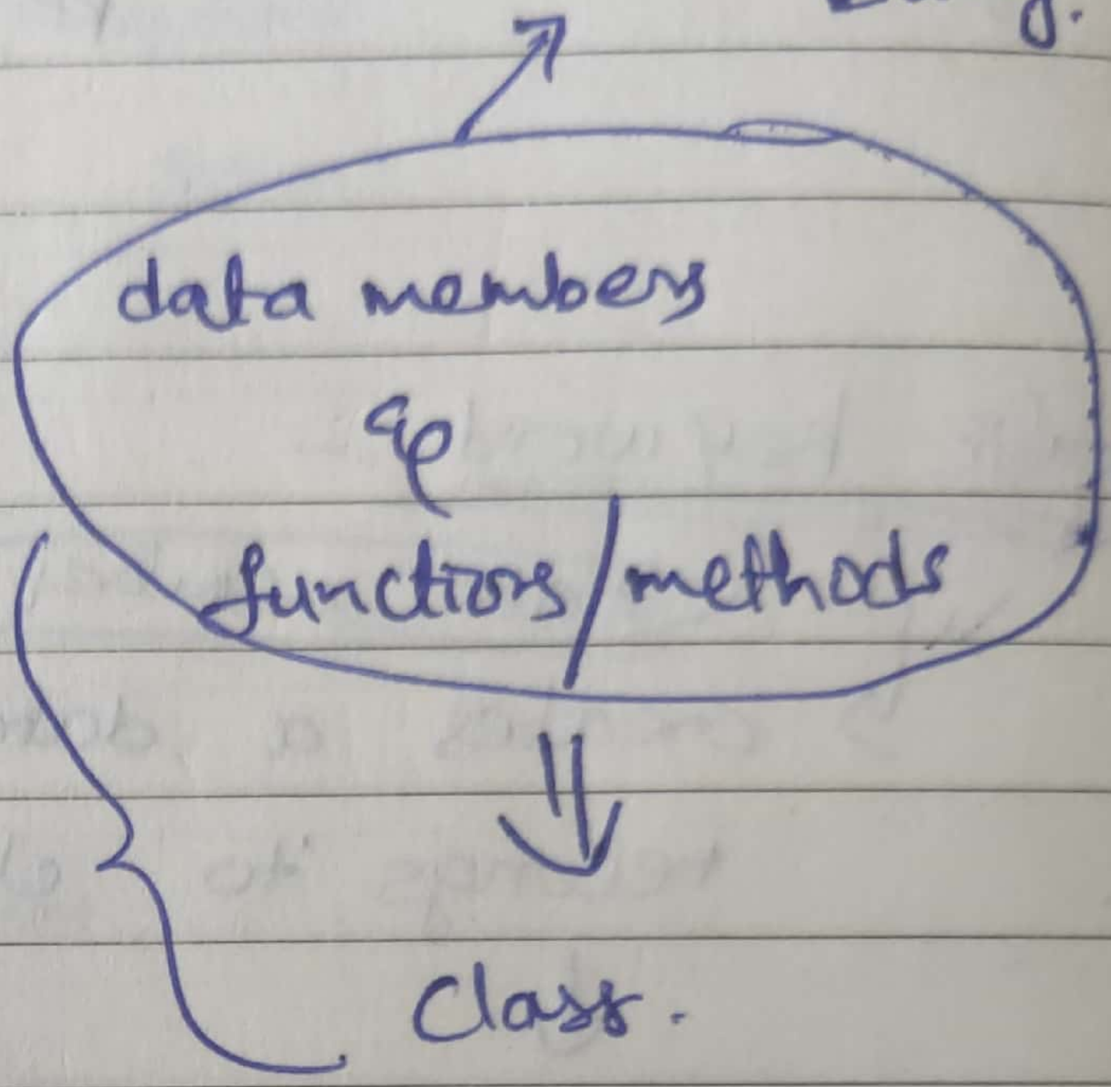
↳ It can only access static members.

It cannot use other data members and throws an errors.

Video 2:- (Love babbar)

Encapsulation:-

↳ wrapping up



Fully Encapsulated Class:-

All data members are 'private'.

Encapsulation:- (Information hiding/ data hiding)

orient electric

DATE / /

Advantages of Encapsulation:-

→ Data hide (Private data members)

↳ security Uses.

→ If we want, we can make class - "Read Only" (by not using setter)

→ Code Reusability.

→ Unit Testing.

Inheritance :-

Inheriting properties or methods from other class.

Ex:- human → is class

↳ height

↳ weight

↳ age.

Male → is a class

Female → also a class.

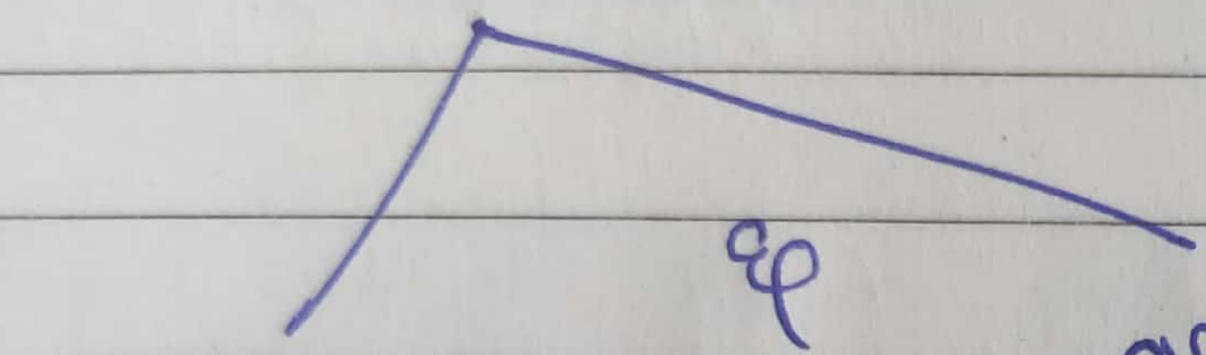
Male and Female are inherited from human class.

Human (Parent class / Super class)

↓

Male (child class / Sub-class)

'Protected' → access modifier



Syntax!-

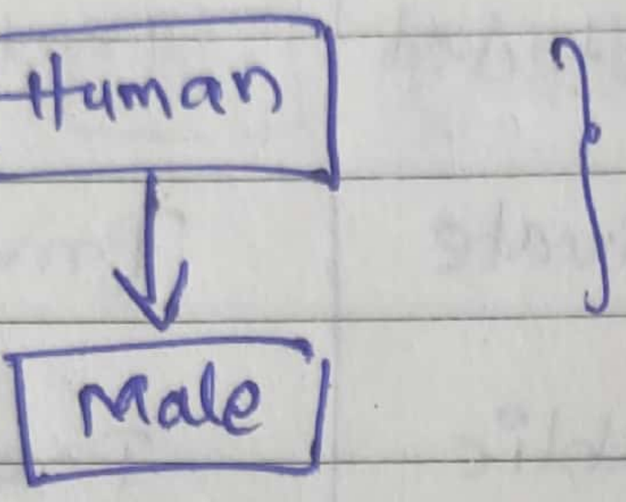
class male : access_modifier Human {

{ ;

Parent	Child	
Public	Public	Public
Public	Protected	Protected
Public	Private	Private.
Protected	Public	Protected
Protected	Protected	Protected
Protected	Private	Private
Private	Public	Not Accessible.
Private	Protected	
Private.	Private	

Types of Inheritance:-

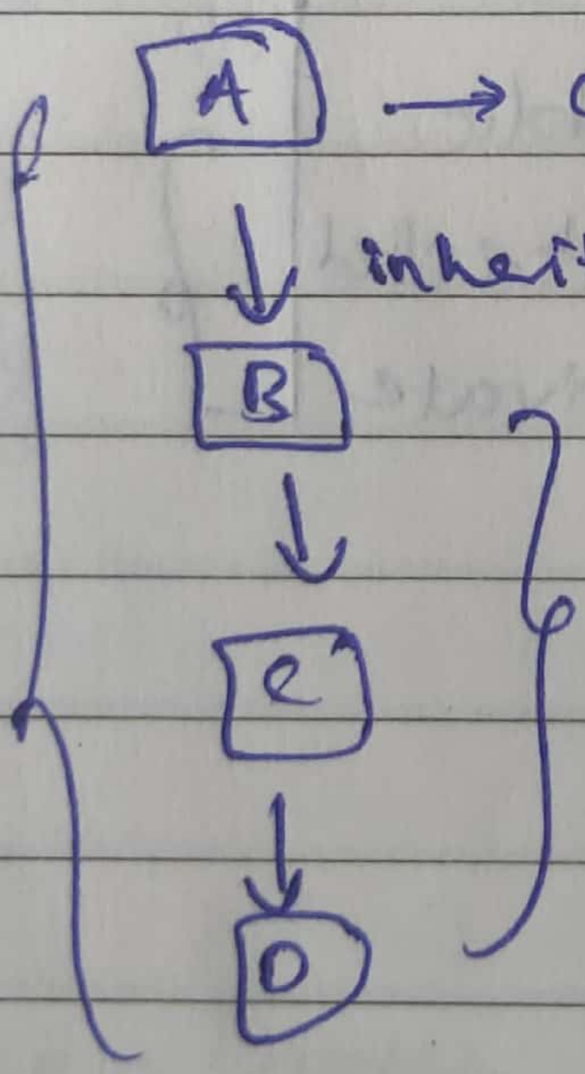
→ Single Inheritance:-



} Single Inheritance.

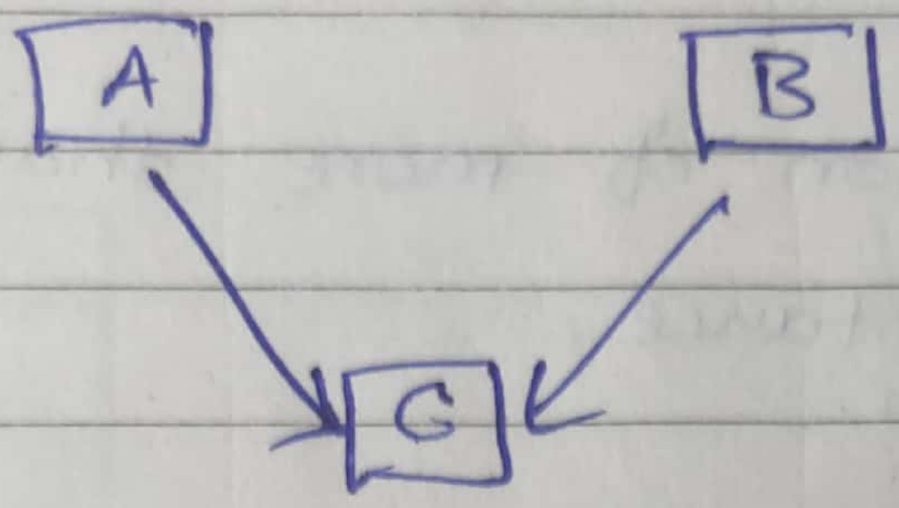
→ Multilevel Inheritance:-

multi-level Inheritance.



diff. levels.

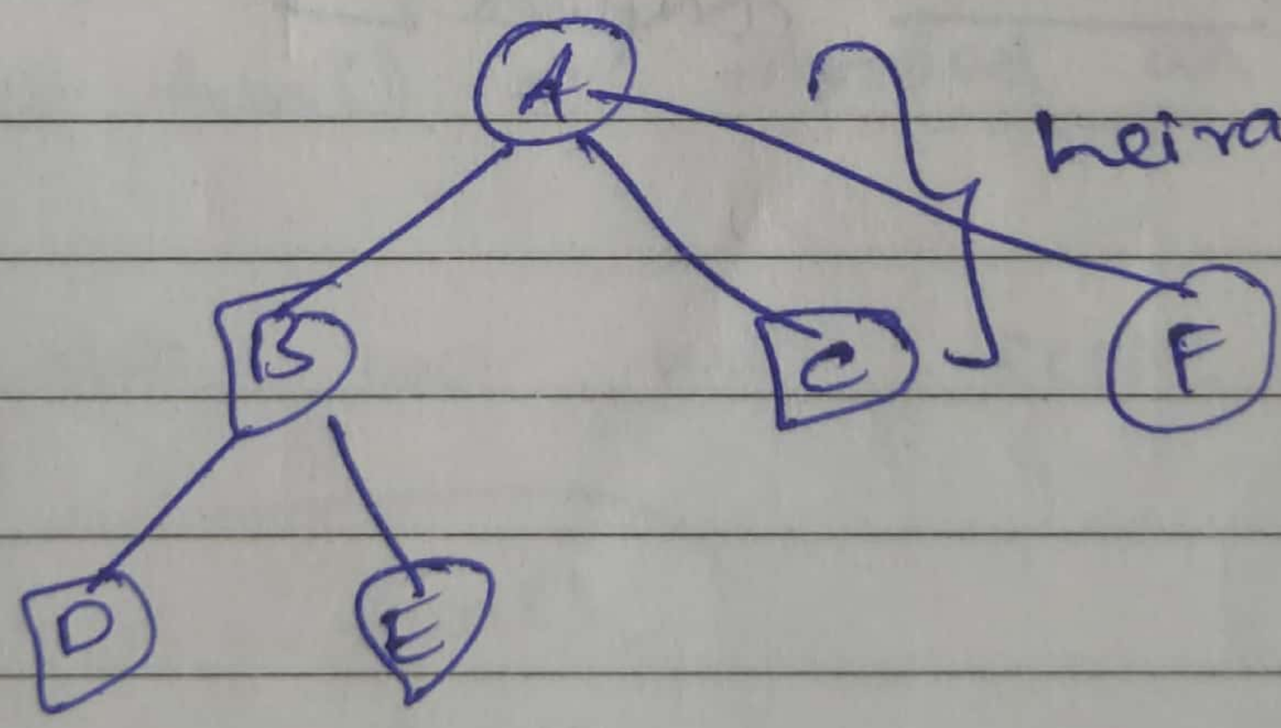
Multiple Inheritance:-



Class C : Public A, Public B {

};

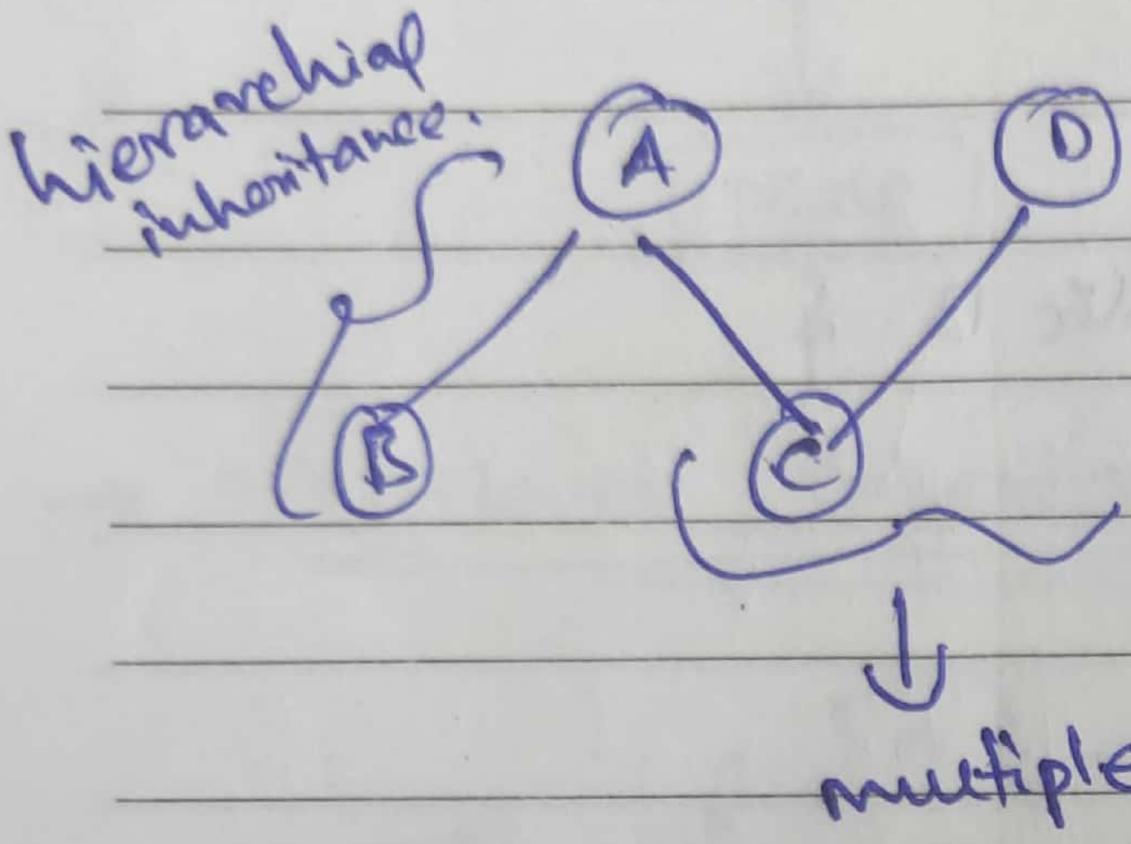
Hierarchial Inheritance:-



One class serve as parent class for more than 1 class.

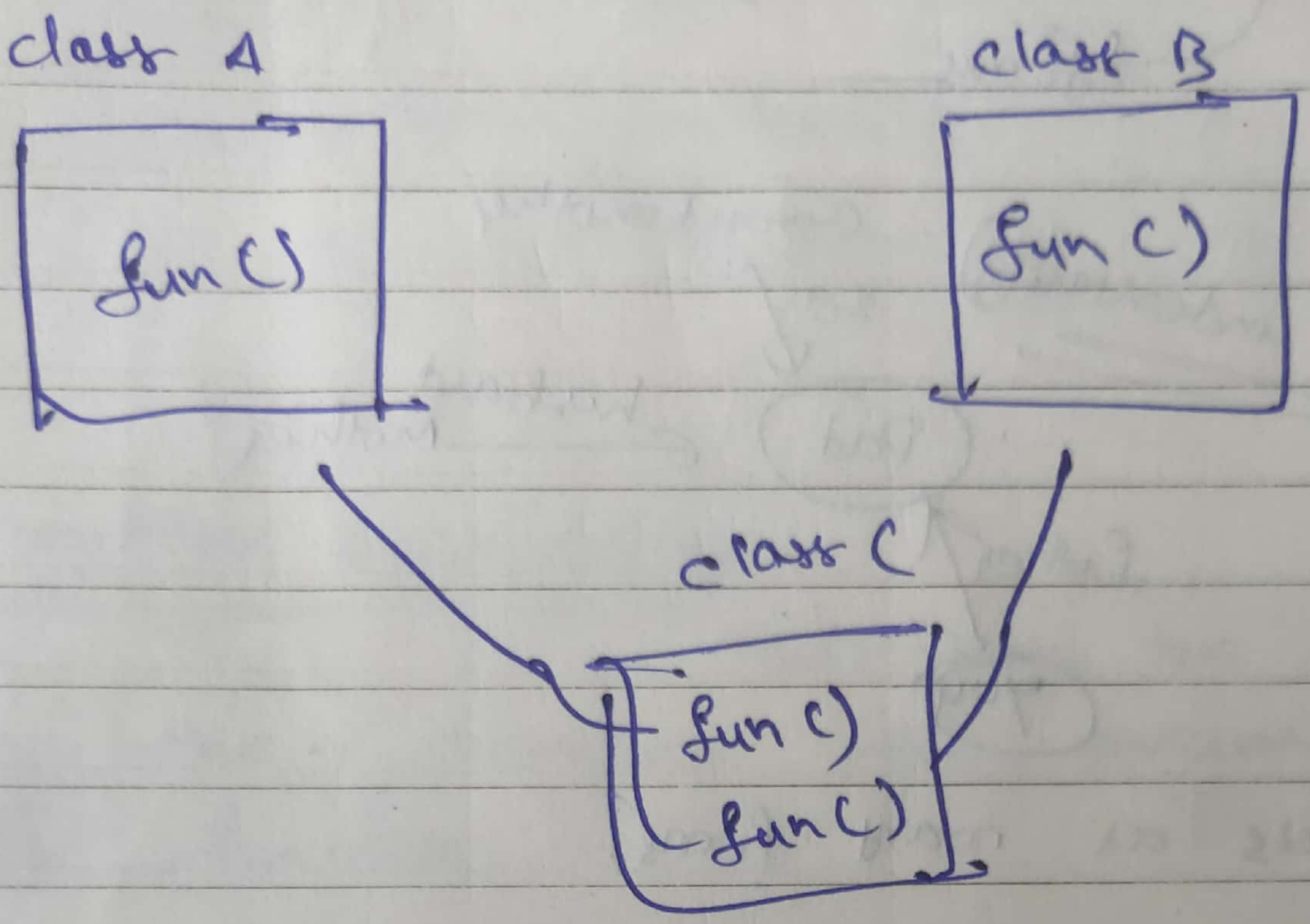
Hybrid Inheritance :-

↳ combination of more than one type of inheritance.



→ completed ←

Inheritance Ambiguity :-



~~c~~ c.object.

(X) object.fun() → throws an error.

To avoid this we use scope resolution.

(✓)
object.A::fun();
Object.B::fun();

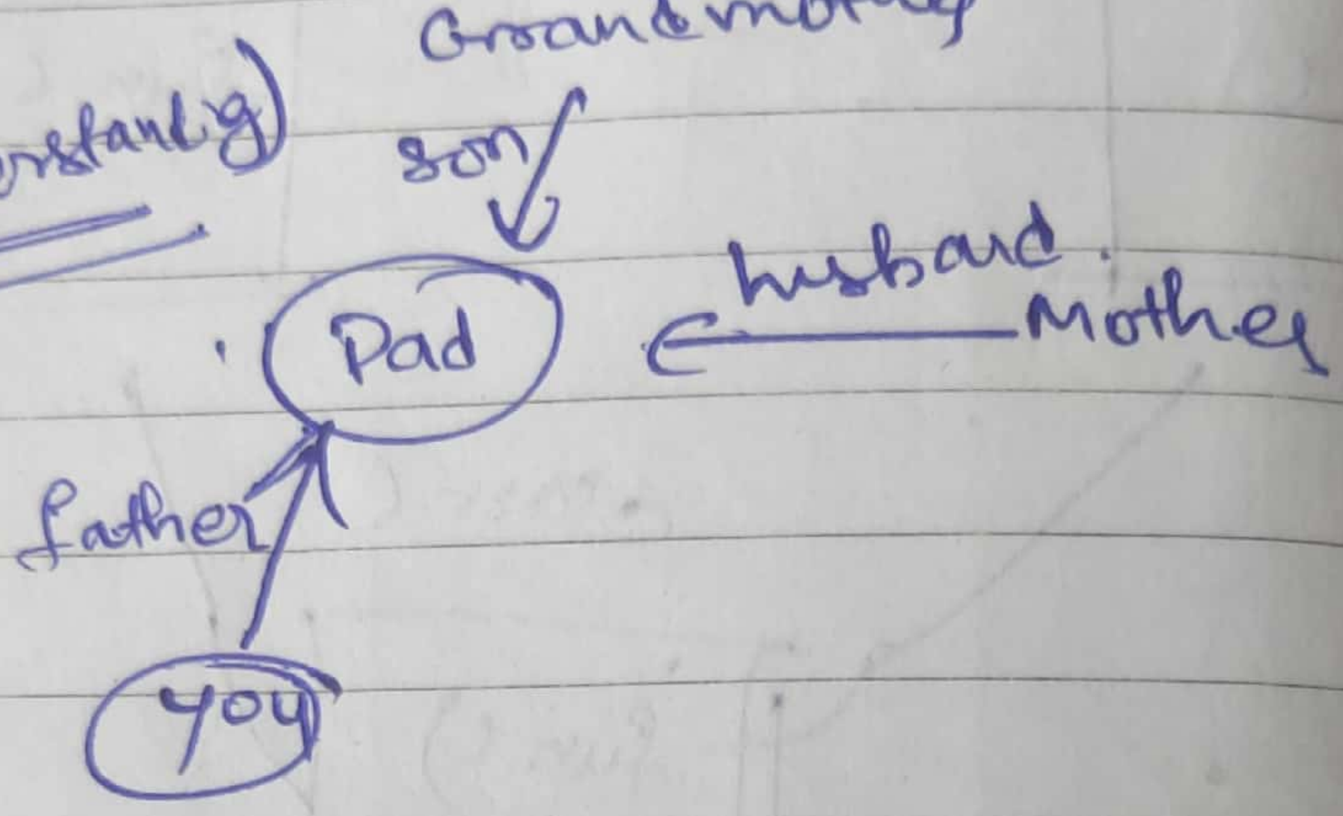
orient electric

DATE

Polymorphism:- (exists in multiple forms)

many → forms.

Ex:- (for understanding)



Dad acts as many forms.

Polymorphism:-

- compile time Polymorphism
- Run-time Polymorphism.

orient electric

DATE / /

Compile time Polymorphism:- (Static Polymorphism)

- Function Overloading
- Operator Overloading.

Function Overloading:- This can be acheived by the changing the arguments of functions.

** Changing return type is not function overloading.

Ex:-

```
void func()
{
}

void fun(int n)
{
}
```

```
void fun(int n, char c)
{
}
```

function overloading

Operator Overloading :-

you can keep any operator.

Syntax :

void.

return-type Operator + ( )

{

cout << " hello " << endl.

a1 + ab ;

}

Run-type Polymorphism (Dynamic polymorphism)

→ Method Overriding.

Inheritance dependent

orient electric

DATE

Method Overriding

class A { → Parent class.
Public:
void speak()
{
cout << "speaking" << endl;
}

E:

class B : Public A → child class.
{
Public
void Speak()
{
cout << "barking" << endl;
}
}

Rules:-

→ method of parent class and child class must have same name.

→ method of parent class and child class must have same parameter.

→ It is possible through inheritance only.

Dog d;

d.speak();

⇓

O/P = barking.

So, this is method overriding.

Abstraction :- ("Implementation Hiding")

This can be achieved by using the access modifiers : Public, Private Protected.