OOPs Class-2

Special class

⇒ ① 8 → 11pm ☆☆ ✓✓
↳ OOPS
concepts ✓✓

✓✓ ② String ⇒ STL using OOPS
☆☆

Love Babbar • Oct 25, 2023

⇒ int a ; int b ;

Student S1 , S2 ; → Ctor

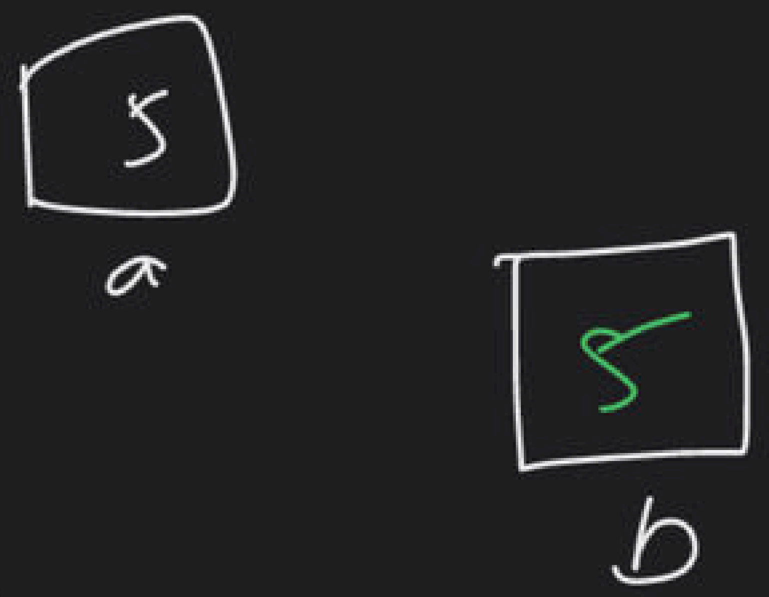
① garbage

② Paramte

⇒ int a = 5 ;

int b ;

b = a ; ?? = Copy



Student s1(- - - - - - - ch);

Student s2;

s2 = s1; ⇒ Copy Ctor

⇒) Q → Student s1; ⇒ By default Compiler add default Ctor.

⇒ S2 = S1; ⇒ Copy ctor define nai kiya

⇒ Compiler's default copy ctor added

⇒ Shallow copy ☆☆

int a = b; → do int copy hota ha,

a ↓ dest   b ↓ src

⇒ Jab main s2 = s1; → Two Students will be copied.

s2 ↓ dest   s1 ↓ src

⇒ Class ⇒

```
Student(const Student & srcobj)  → s1
{
    this→name = srobj.name;
    this→id = srobj.id;
    this→gf = srobj.gf;
}
          ⋮
```

Why we need copy ctor?

⇒ deep copy krni ho...

☆☆

① Copy.

② deep copy

☆

⇒ Shallow vs Deep copy

⇒

```
main( )
{  int a;
    a = 5;

}
```

Why we need copy ctor?

Shallow vs

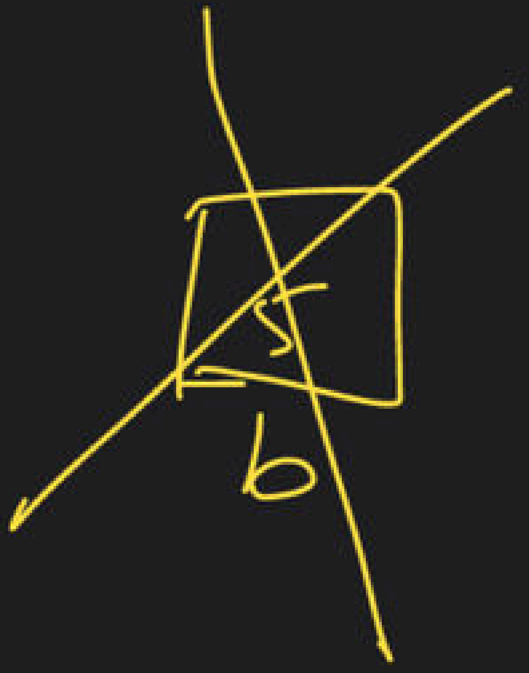
☆☆ deep copy

=> main( )

```
{ int a;
  fun( )
  a = 5;

  return 0;
}
```

```
void fun ( )
{ int b;
  b = 5
  return;
}
```



=> Variable life cycle

1. init
2. copy
3. destroy

Life cycle of an object

=>

```
main()
{
    Student S1(------); → ctor

    return 0; → S1 will be destroyed;
}
```

S1 X

S1 will be destroyed;

⇒ ① ctor ✓

② dtor → destructor ✓

⇒ ab tak mene dtor banaya ??

⇒ if you don't write dtor, compiler will take the responsibility

2 min paani Break

lakshayk12

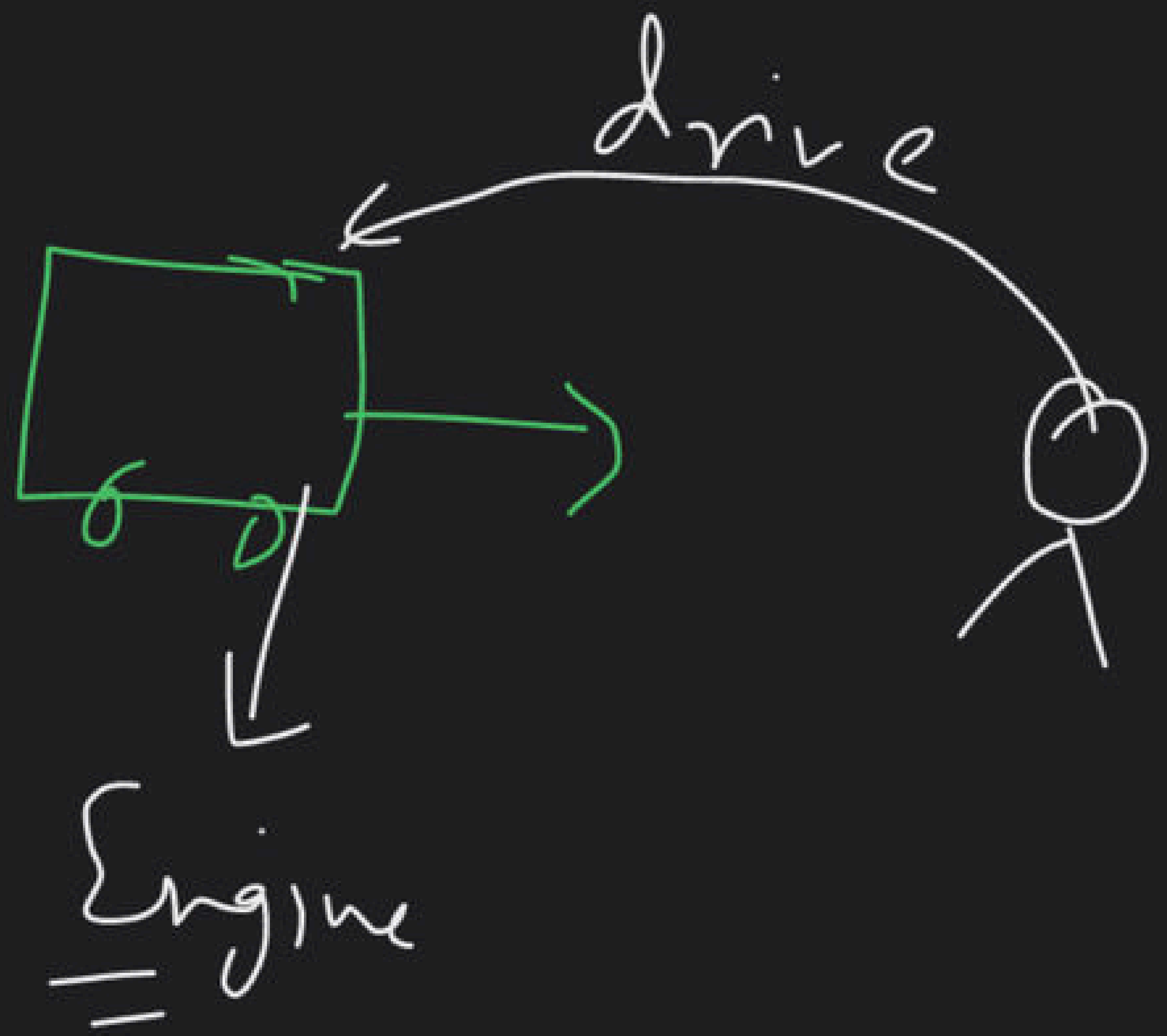
↓

Insta ID

⇒ ① pillar of oops

↳ Abstraction ★★★ → loose coupli

Coupling



⇒ ② phone

↳ Screen ✓

↳ Touch

→ Jeevan me agr Abstraction To jeevan aasan hoga ☆☆

→ Aap chizo ko me krte ho & rakh dete ho.

Abstract ☆☆

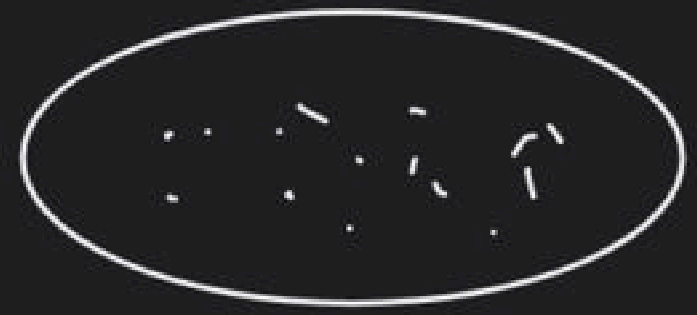
=> #include<string>

```
string a;          ⟶   a => 'b'

a.push_back('b');
```

① Encapsulation ☆☆ → is a way to implement abstraction

→



→ WRAP

⇒ Bundling of data & Methods

Student S1;

Stu S2;

S2. —— ()

S2. Study();

⇒ Why Encapusation?

① Easy to handle

② Protect Integrity (Security)

↳ control how class data is modified.

friend keyword ☆☆☆

③ Maintanaibilty ☆☆

=> Perfect Encapsulation

- if all datamember are Private
- Through getter/selter.

⇒ Inheritence

⇒

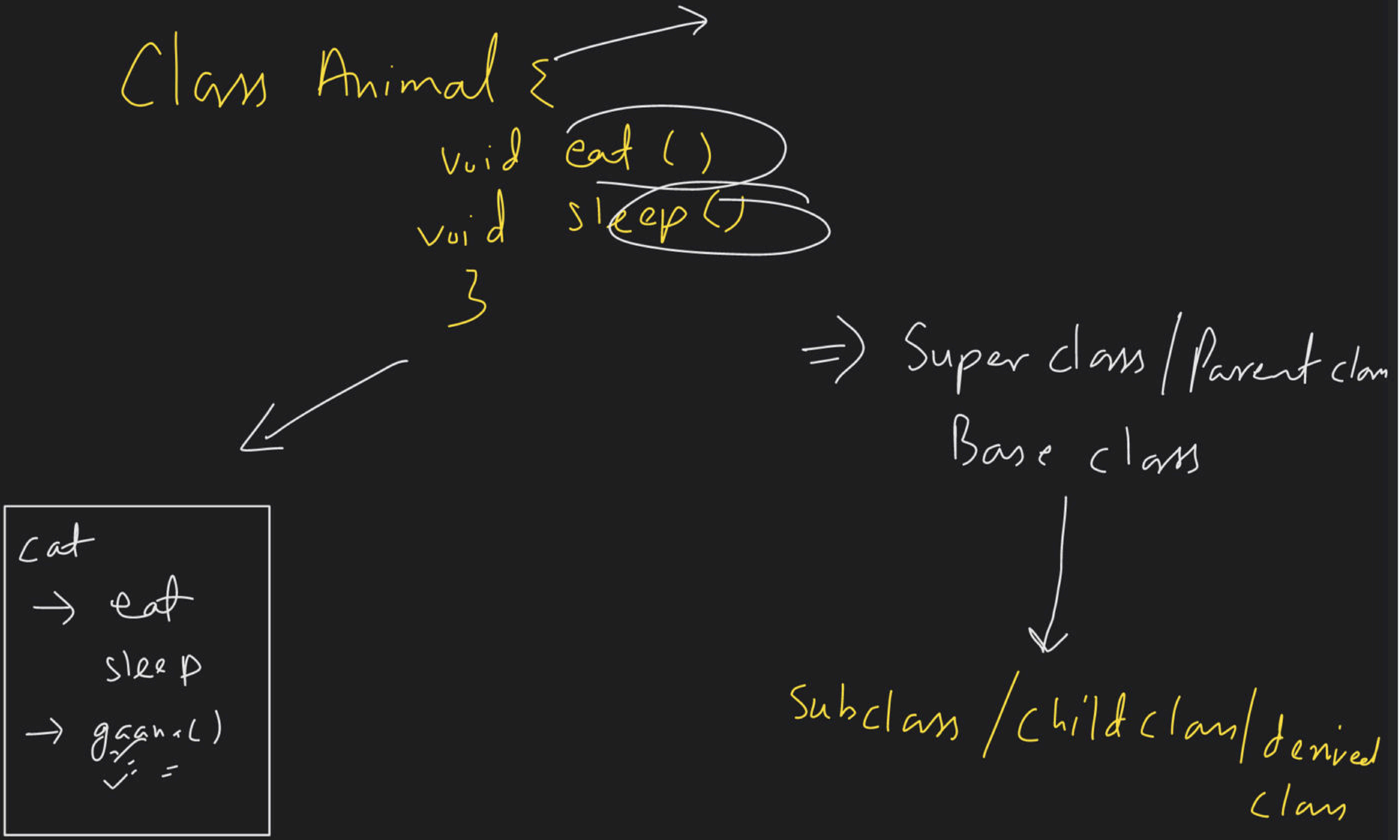
⇒ Persian cat → Ma → eyes orange

eye orange

Parent → attributes

↓

Son → attributs ✓

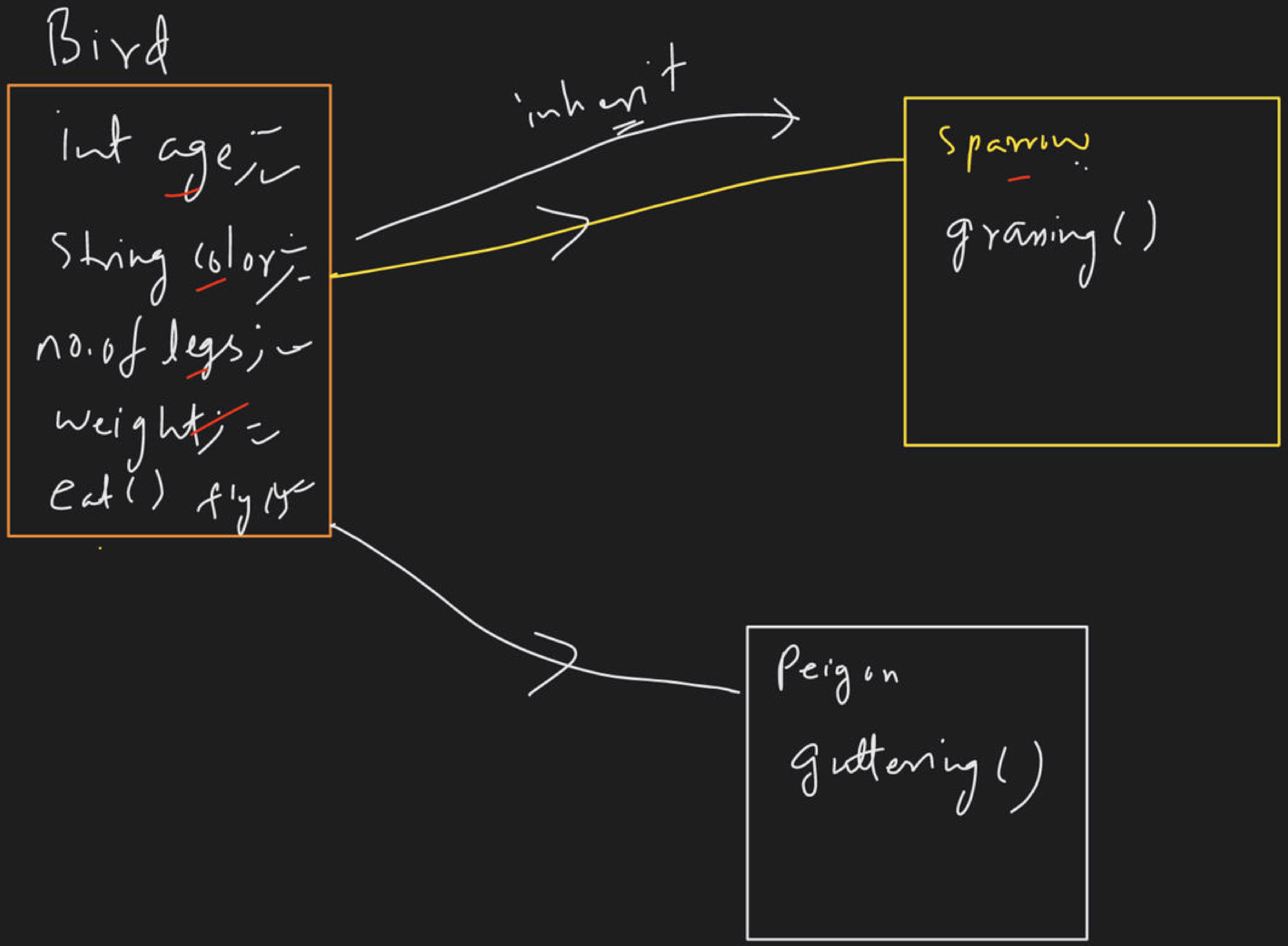


```
Class Animal {
        void eat ( )
       void  sleep ( )
       }
```

cat
→ eat
sleep
→ gaana( )

⇒ Super class / Parent class
Base class

Subclass / child class / derived class


Bird
int age;
String color;
no.of legs;
weight;
eat() fly
inherit
Sparrow
graning()
Peigon
guttering()

=) Syntax

```
class Child_name : Pubic ---- Parent_name {

};
```

Mode of Inheritence

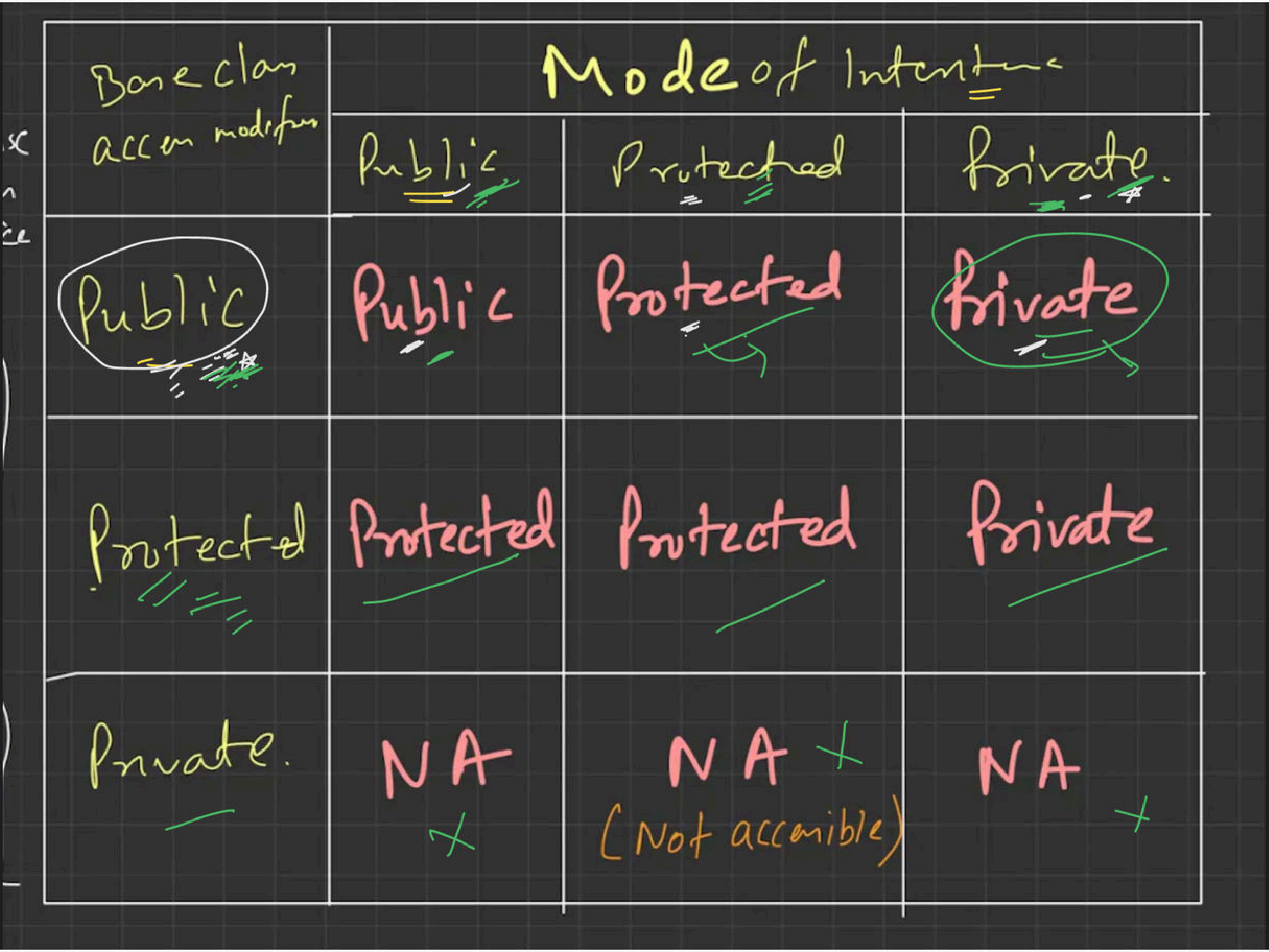
1. Public
2. Private.
3. Protected

⇒ Protected ⇒ Members declared Protected are accessible within class itself & to its derived class.

Private ⇒ within class hota hai

⇒ derived class visibility Not Possible

⇒ Private data can't inherit. ✯

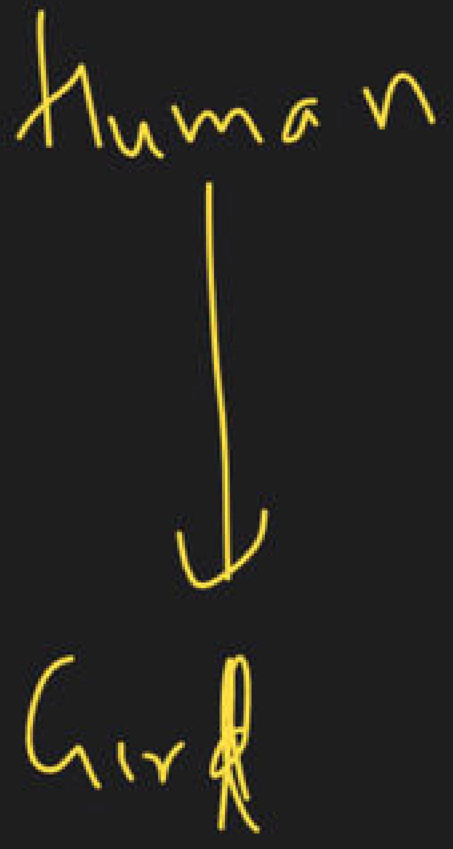


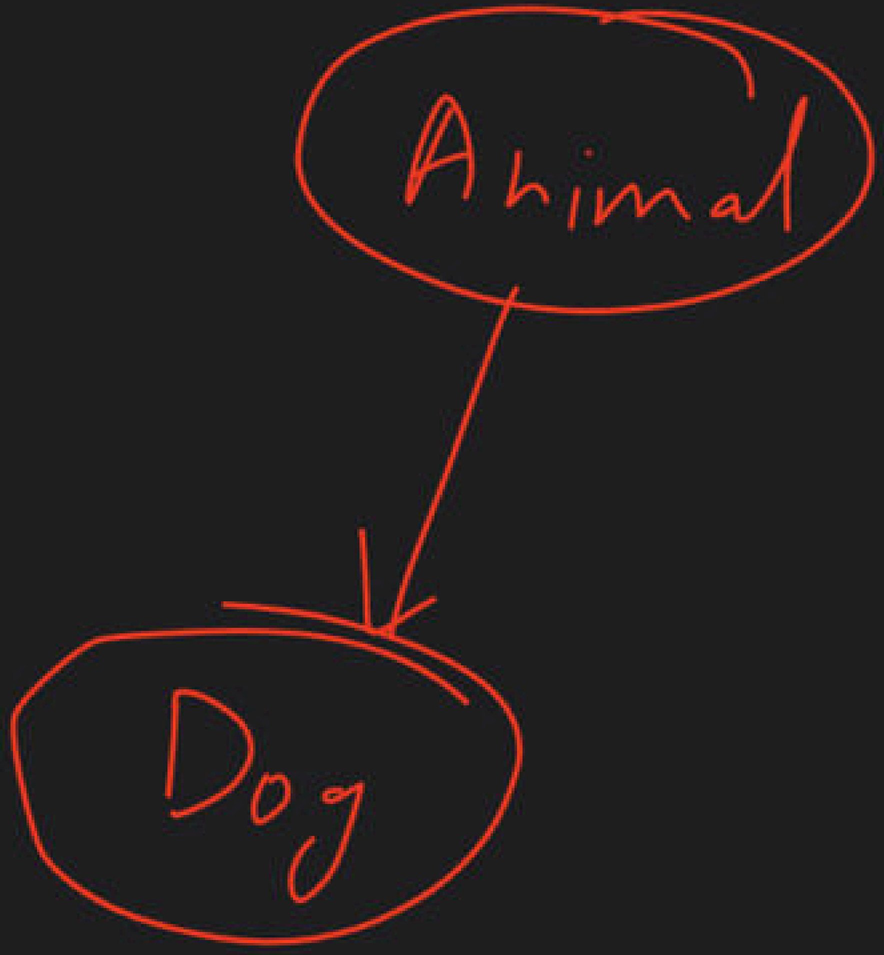
Base class access modifier	Mode of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private.	NA	NA (Not accessible)	NA

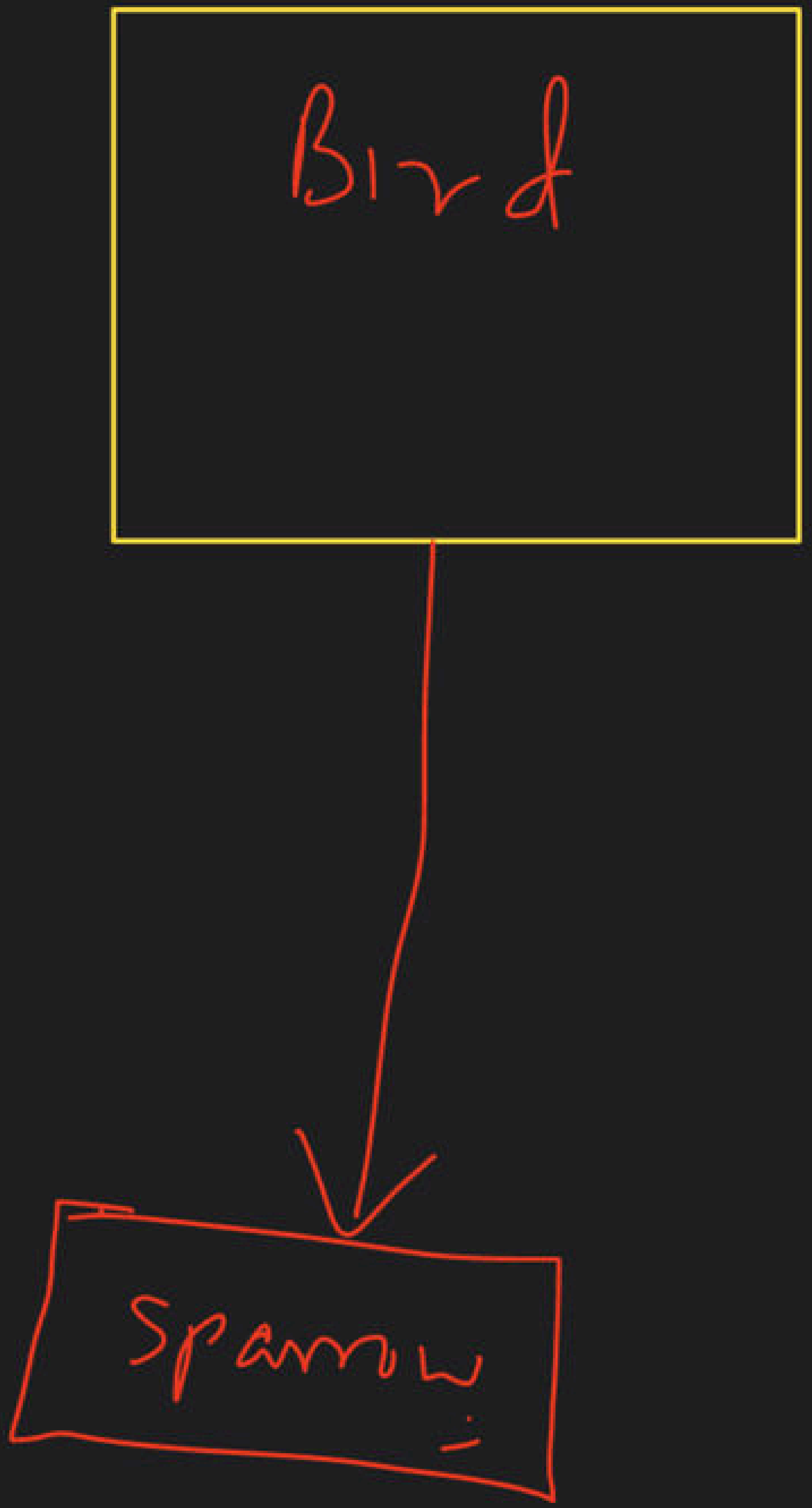
Types of Inheritence

⇒ 'is-a'

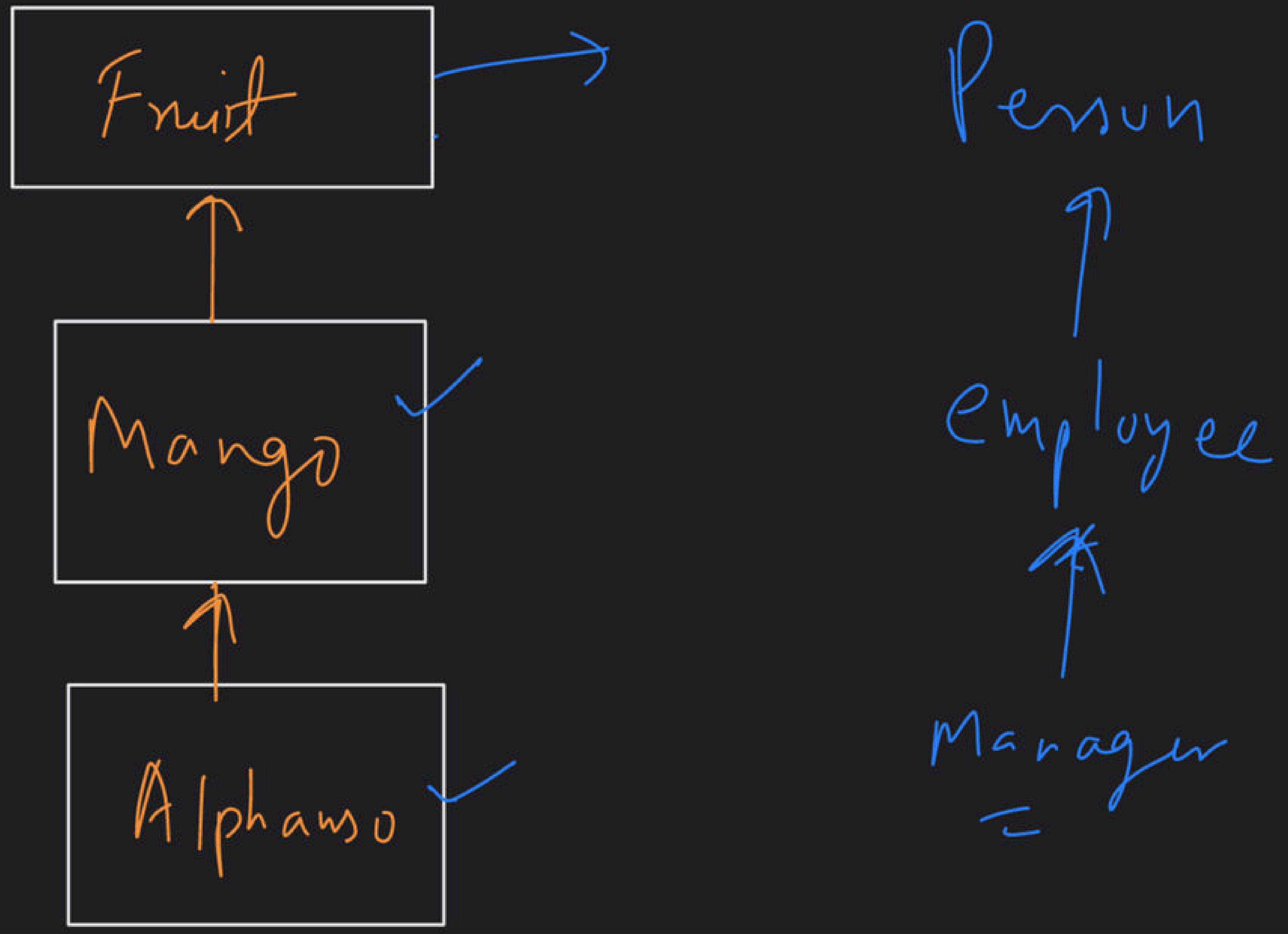
① Single ⇒





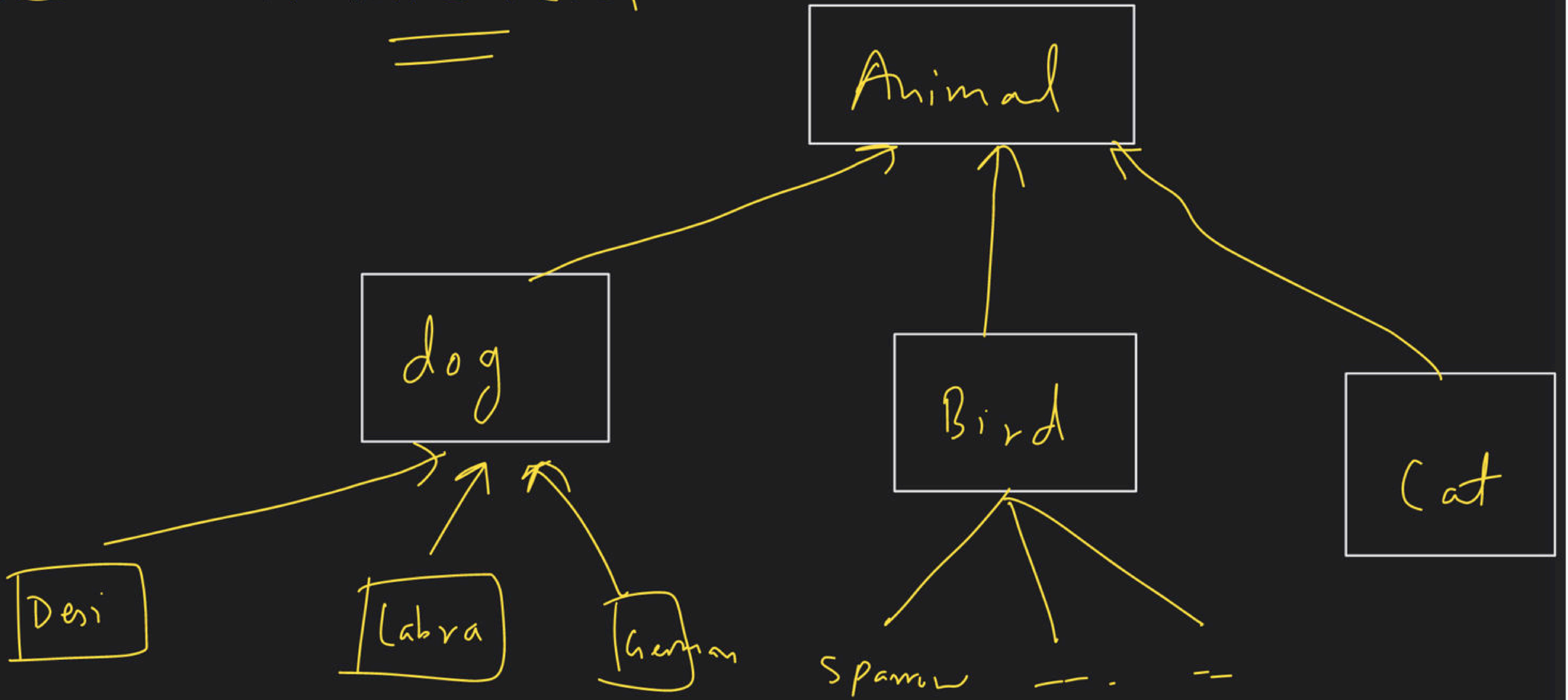


② Multi-Level (chain of Inheritence)



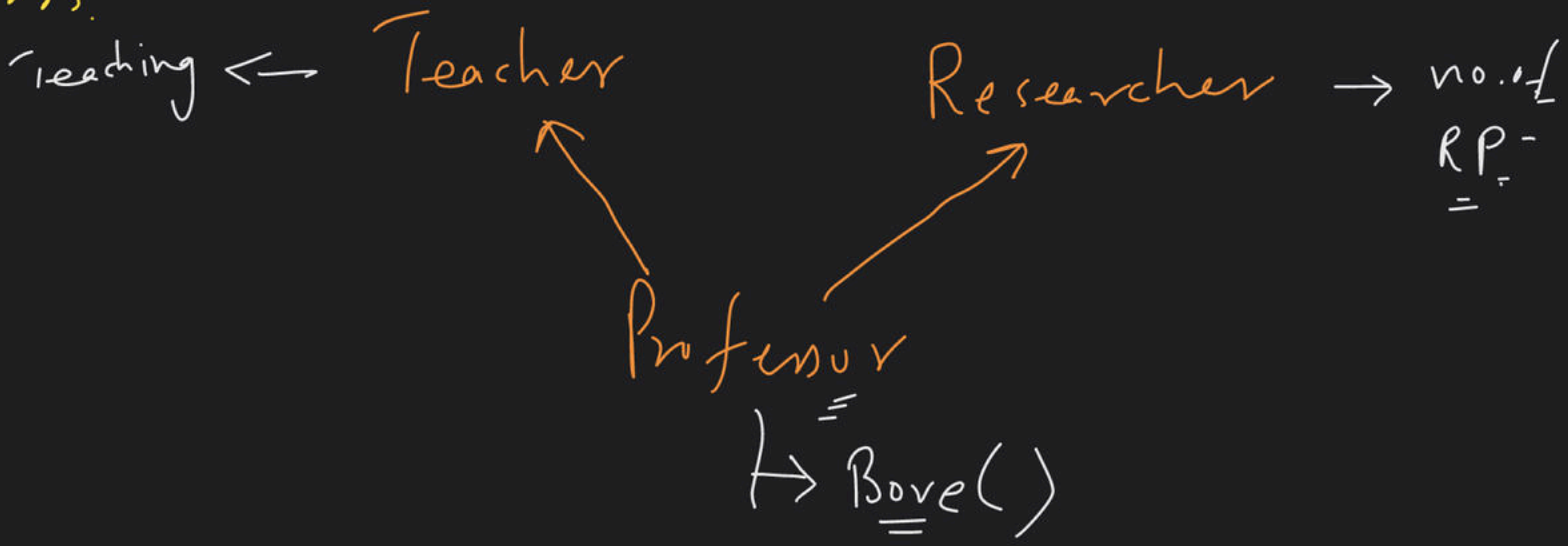
⇒ 3

Hierarchical

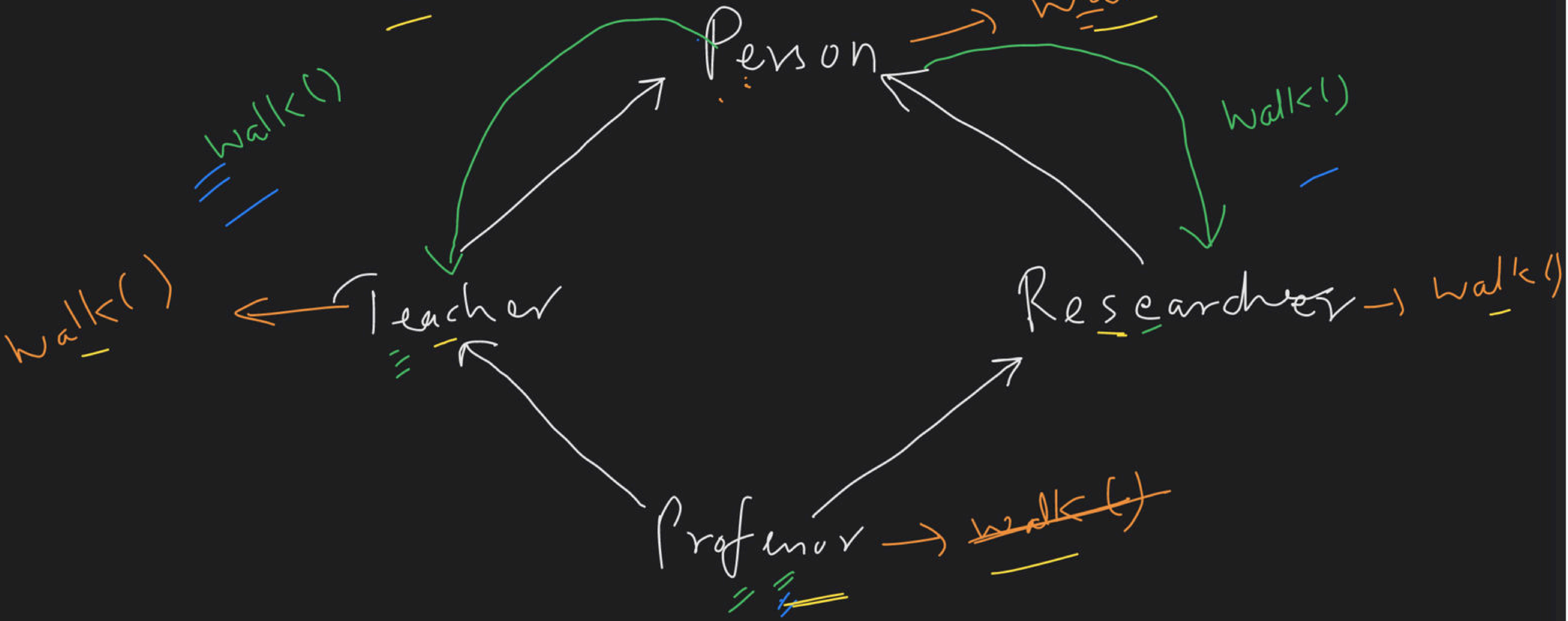


④ Multiple Inheritance (Not Possible ☆ Java ☆)

⇒ Derived class inherits from more than 1 class.



⇒ Diamond_Problem



⇒ #3 Polymorphism

→ Many forms

→ Existing in many form

① Compile Time

② Runtime

Faster Compile Time

⇒ ① Compile Time → (Static Polymorphism)

① f^n overloading → Parametrized ctor

└→ f^n

② operator overloading

② Operator overloading

+, −, ++, --, ,, ( ) . . . .) internet (find all c++ op. which can be overloaded!

''

+ →

⇒

Vector

$$\underset{V_1}{\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}} \quad \underset{V_2}{\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}} \Rightarrow \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

1st ✓

V1 + V2;

same → V1.add(V2);

arg ⇒ V1

dest

Src → V2

```
void operator+(const Vector& src)
{
    this → V1

    this→x = this→x + src.x;
    " → y = " → y + src.y;
}
```

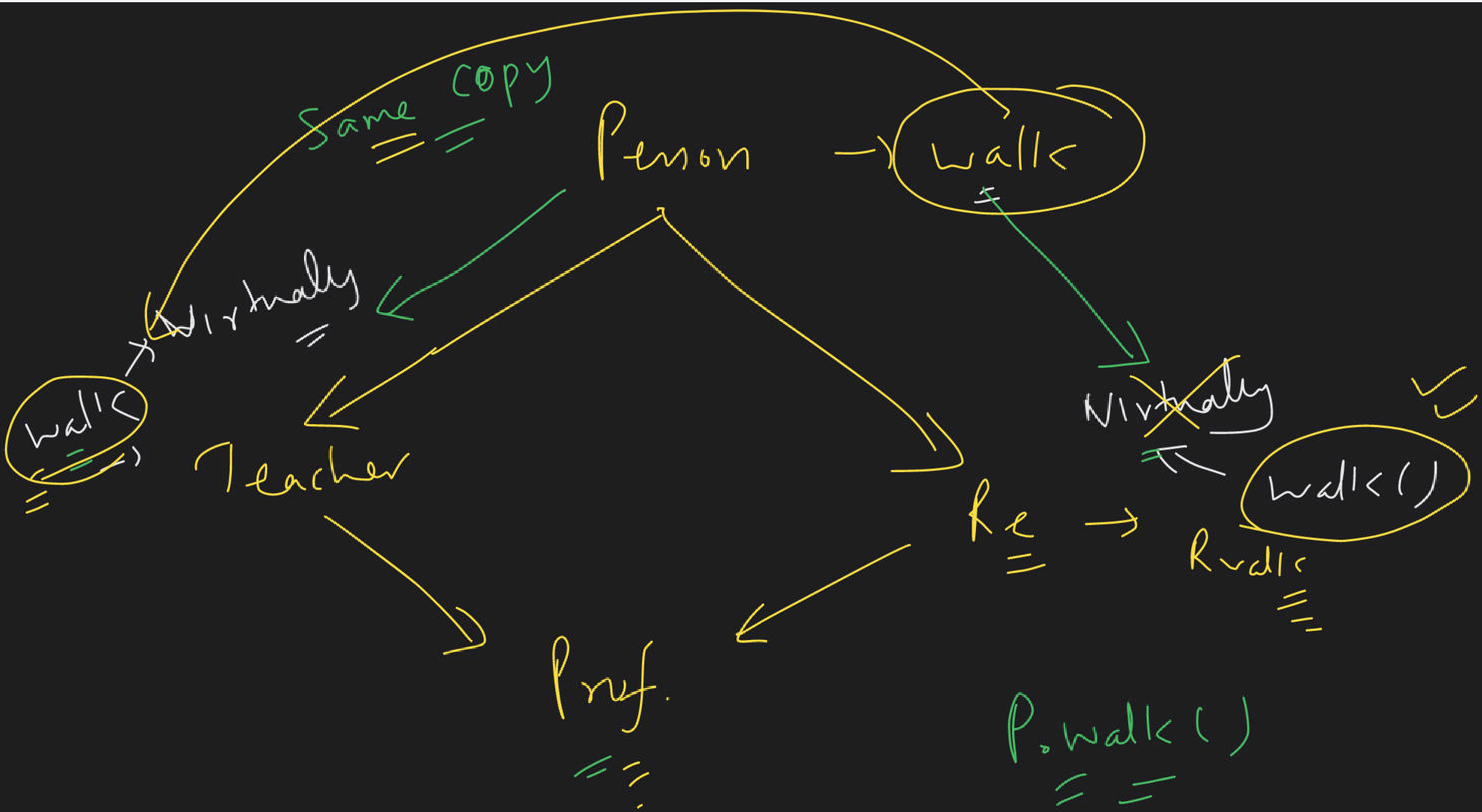
→ V2

src = V2

Sol^n

1. Scope Resolution

2. Using Virtual


Same copy
Person
walk
Virtualy
walk
Teacher
~~Virtualy~~
walk()
Re
Rwalk
Prof.
P.walk()