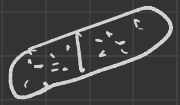
OOPs { Day-2 }

① Encapsulation :-

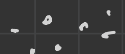
→ "Information Hiding" / "Data Hiding"

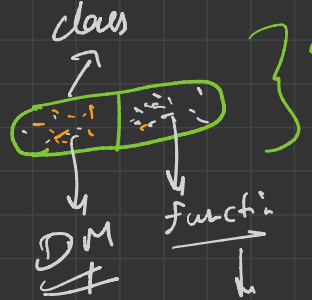
→ Encapsulation

→ wrapping Up Data Member & Function



Data Member → 

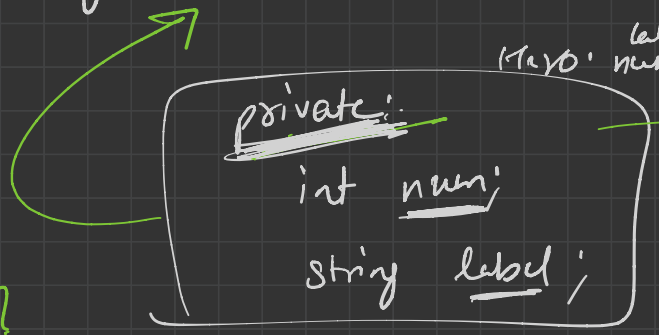
Function → 



↓ property / state

↓ Method / Behaviour

→ Fully encapsulated Class :- all → D.M → private



→ same class me hi access kar paenge

why ⇒ ?

Adv :-

↳ Data Hide → Security ↑

↳ if we want, we can make class - "Read Only"

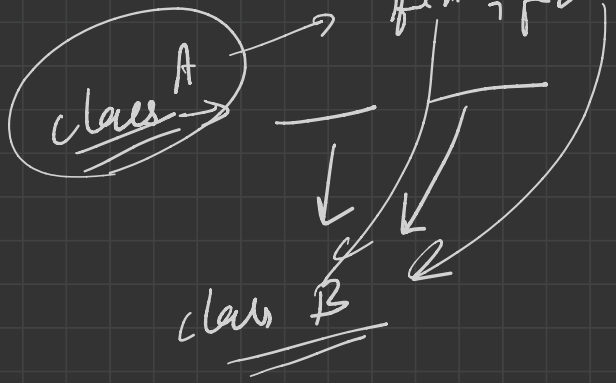
↳ Code Reusability

↳ Encapsulation → Unit Testing

Implement:-

Inheritance :-

→ what -> ?



Ex:-

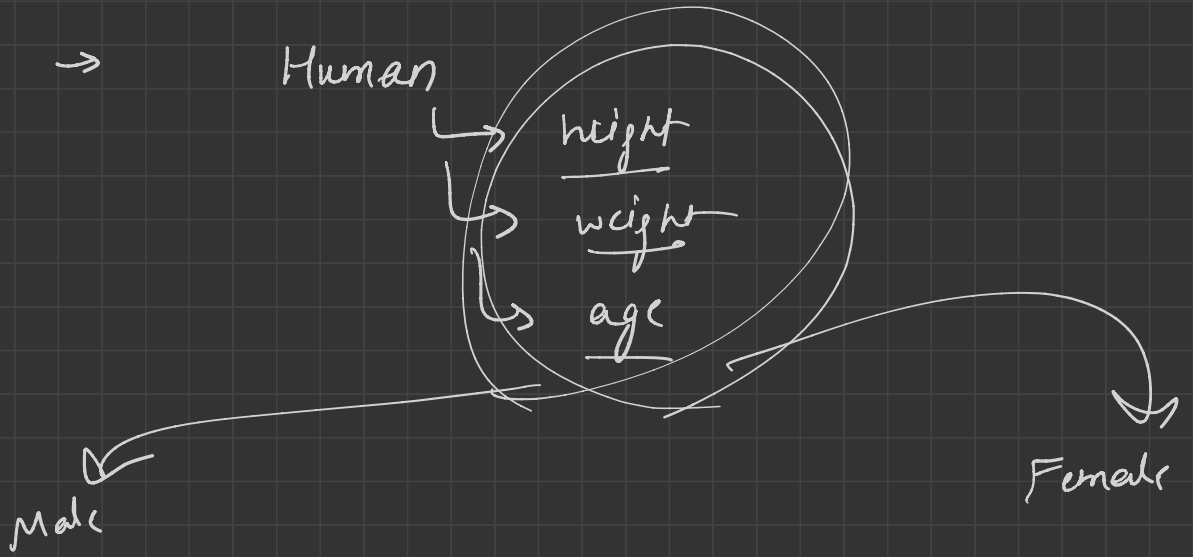
height

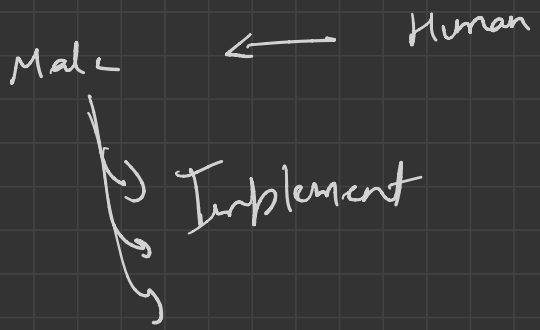
tall

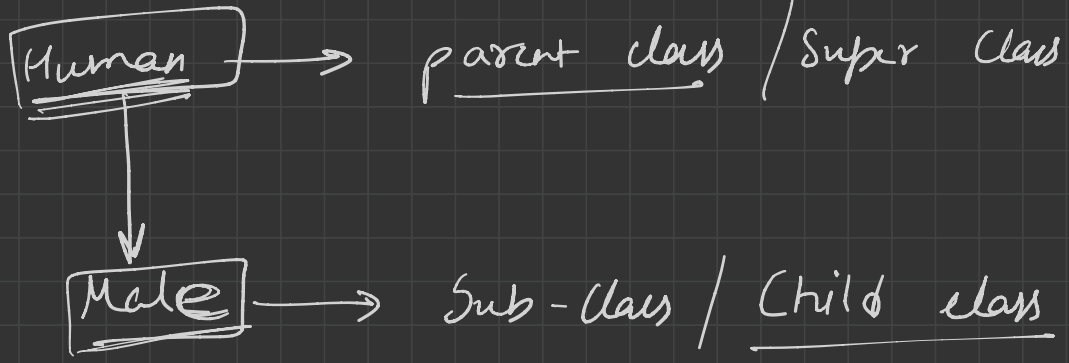
tall

mummy → Silky

Silky


Human
height
weight
age
Male
Female

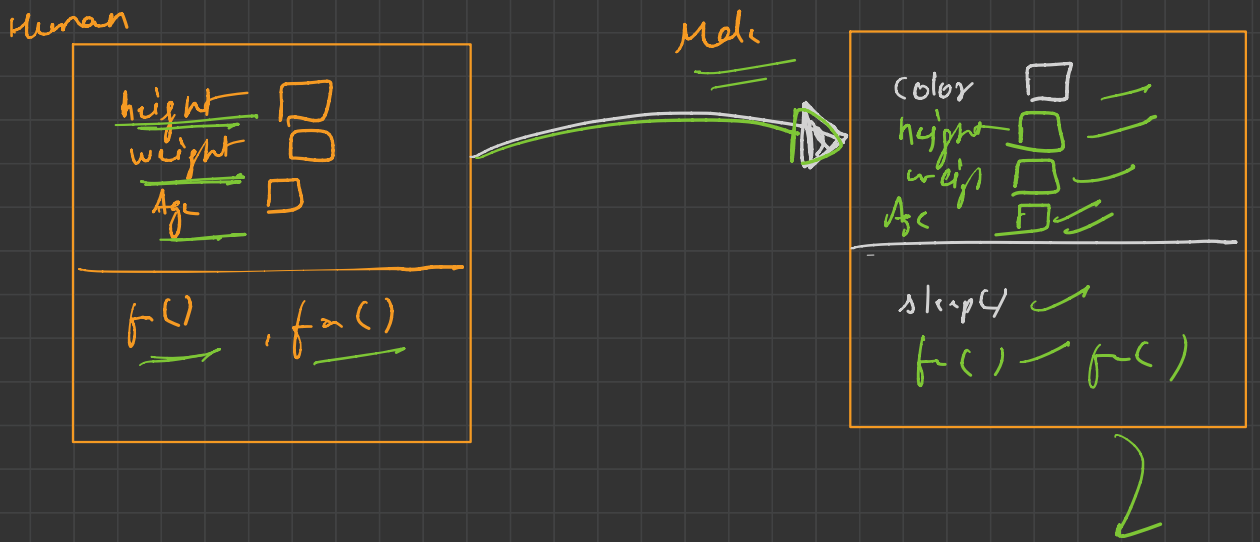

Male
Human
Implement



public mode: -

class ____ : ____ ____

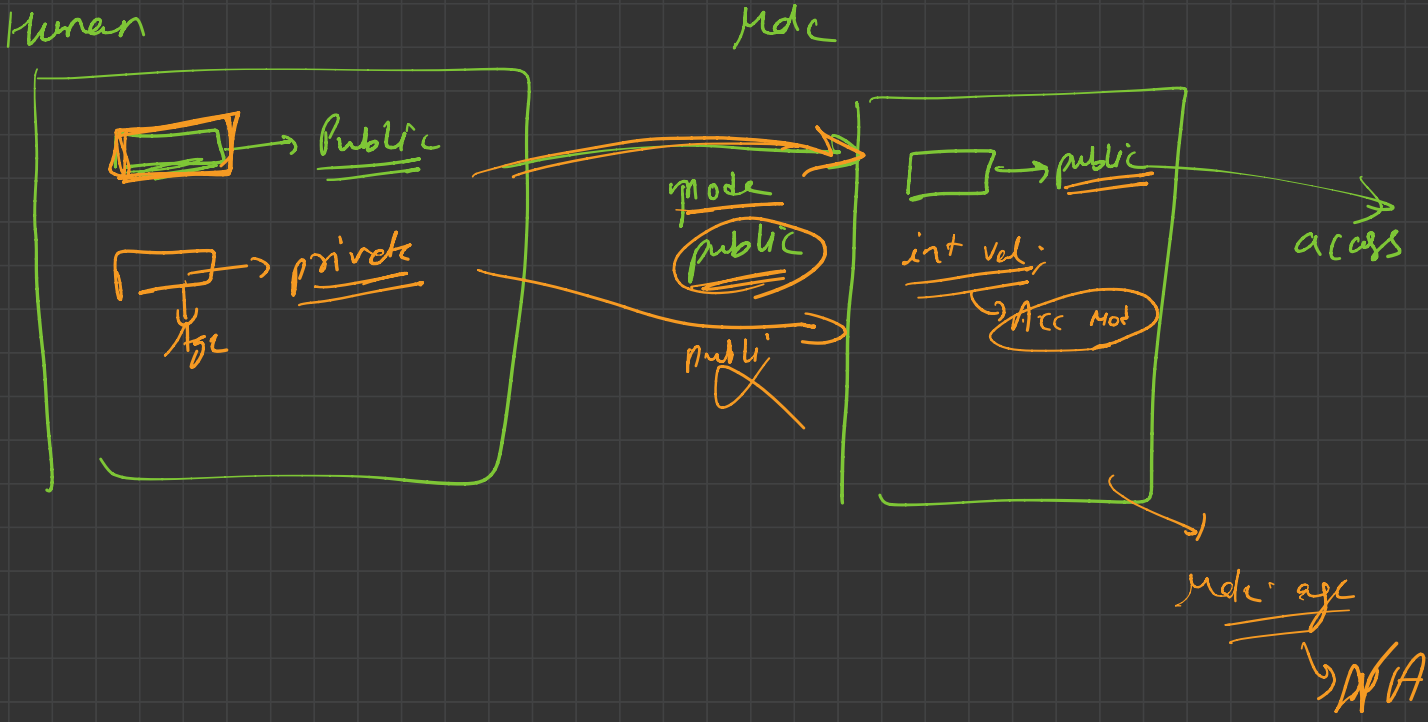
name child — mode — name parent class



male.age

mode of Inheritance

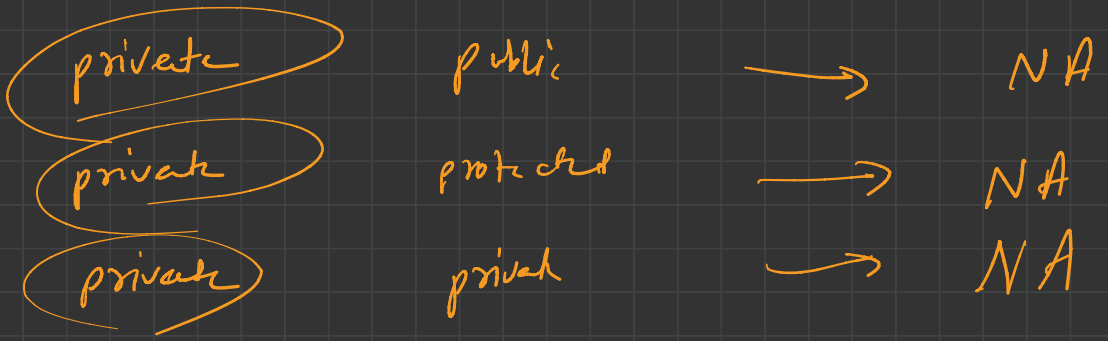
public → inherited

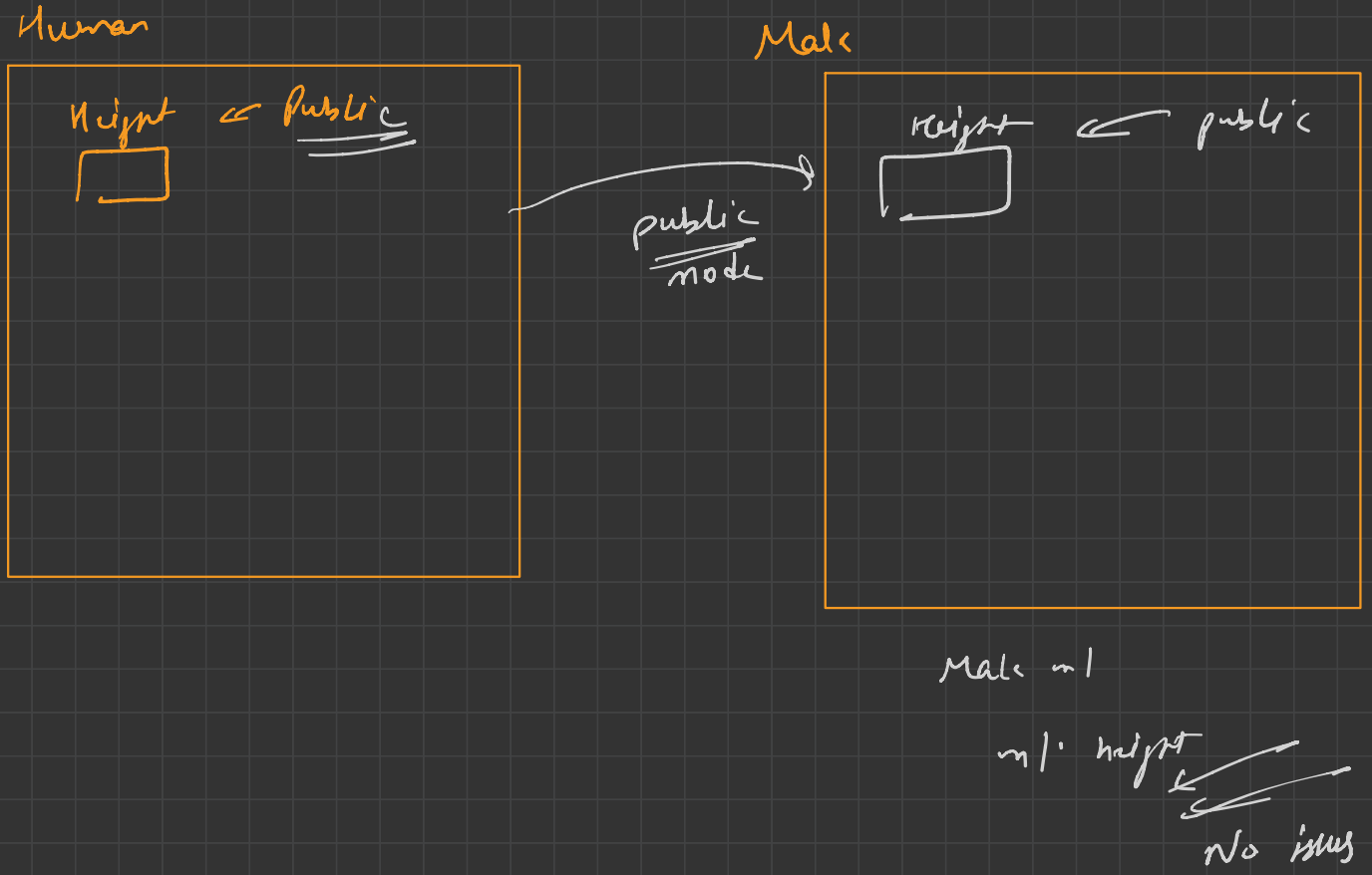

Human
Public
private
age
Mdc
mode
public
public
public
int val;
Acc Mod
access
Mdc: age
NA

Super Class → Access modifier property Data member

Sub Class → Mode of Inheritance

public	public	public
public	private	private
public	protected	protected
protected	public	protected
protected	protected	protected
protected	private	private


private
public
NA
private
protected
NA
private
private
NA


Human
Height
Public
public
mode
Male
Height
public
Male m1
m1. Height
No issues

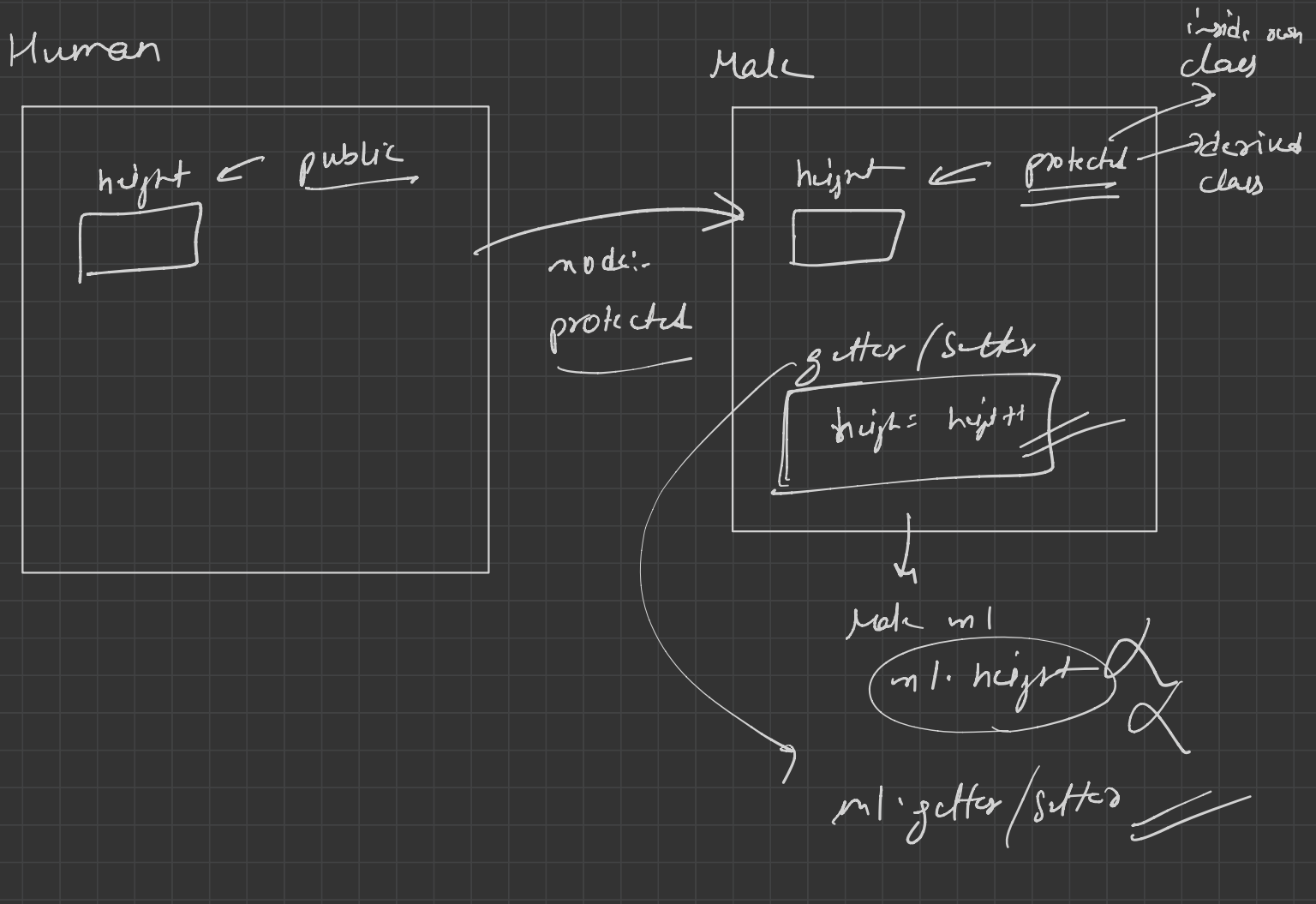
→ protected → similar Private

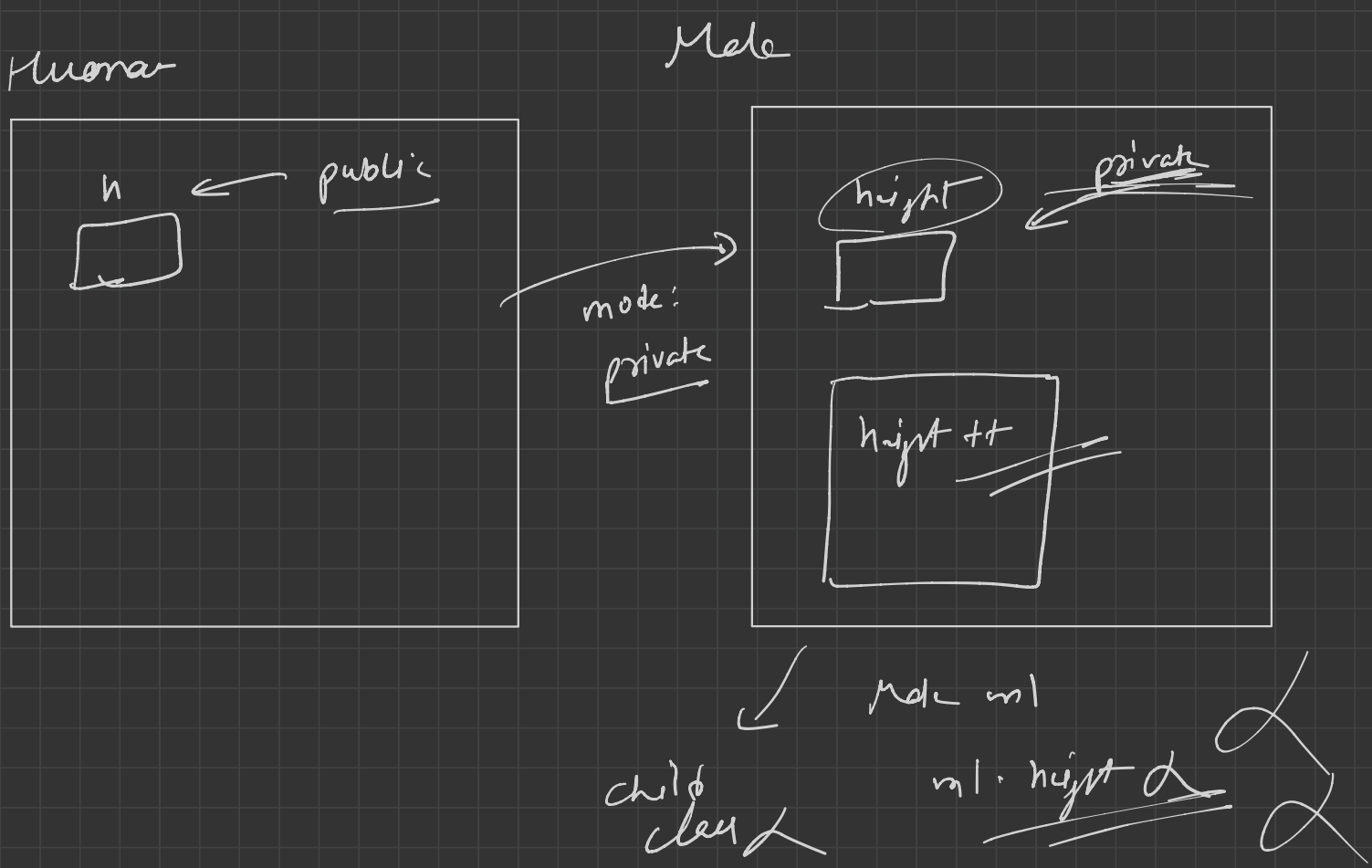
→ 2nd ←

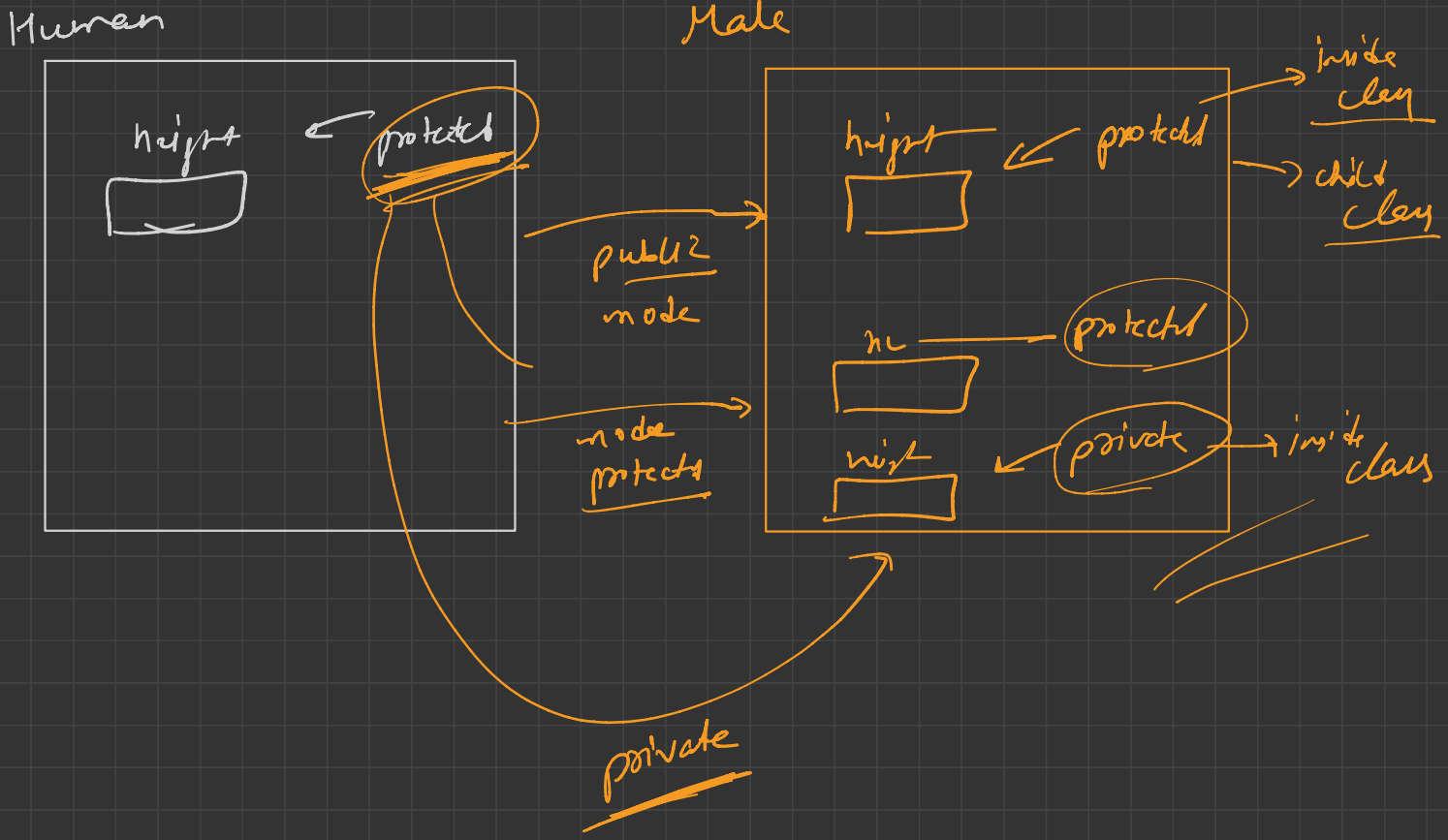
→ child class / Derived class

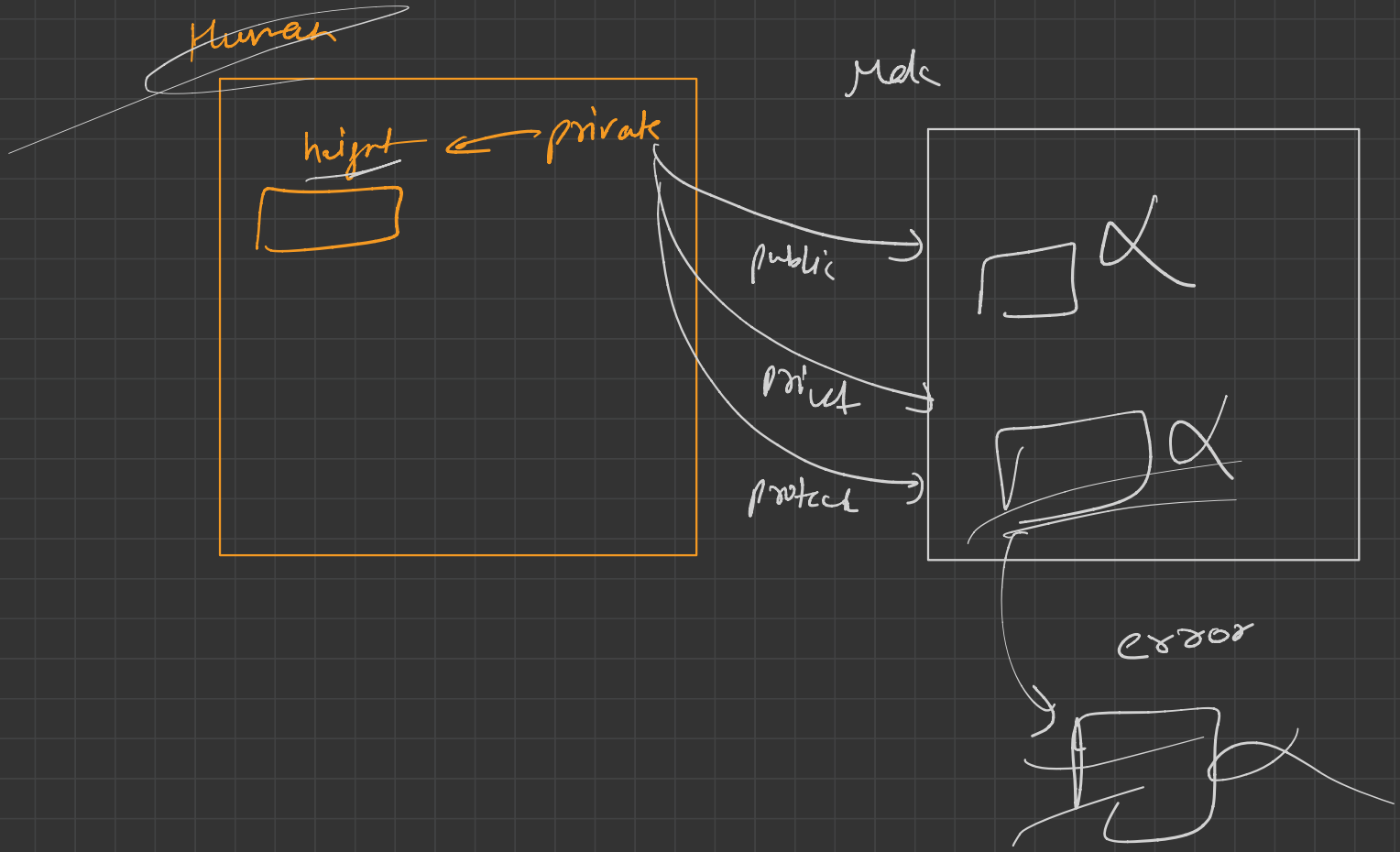
→ can access

accessible inside own class

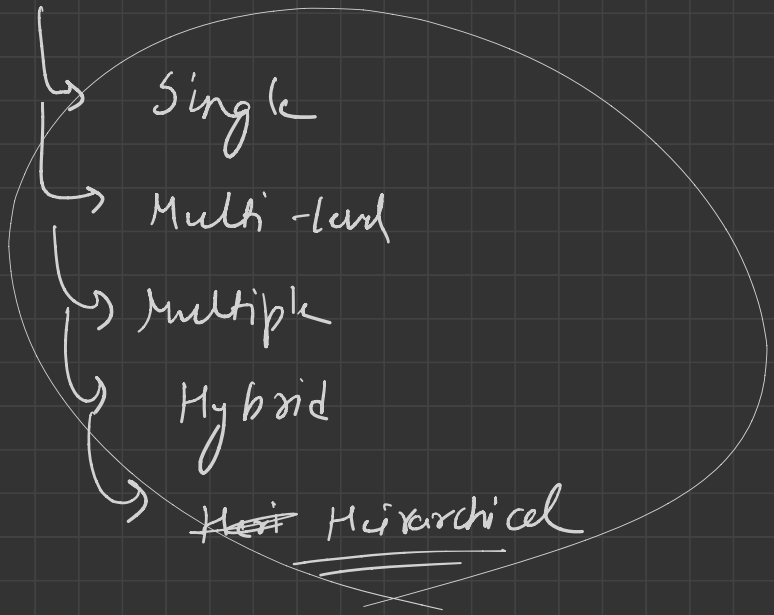

Human
height
public
mode:
protected
Male
height
protected
inside own class
derived class
getter / setter
height = height + 1
Male m1
m1. height
m1. getter / setter


Human
n
public
Male
height
private
mode:
private
height ++
Male m1
child
class
m1. height

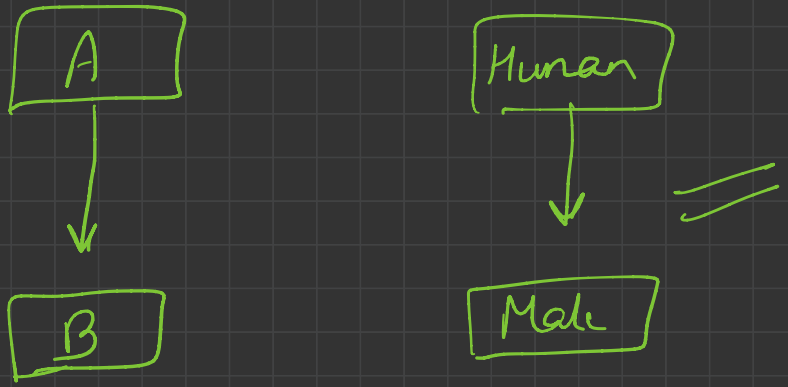

Human
height
protected
Male
height
protected
inside class
child class
public mode
protected mode
private
height
protected
private
inside class
private

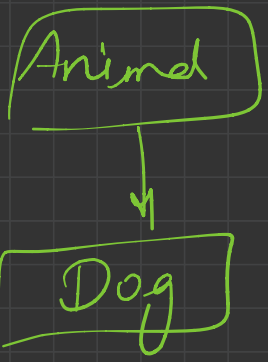

Human
height
private
Male
public
private
protected
error

→ Types of Inheritance:-

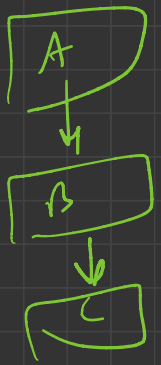


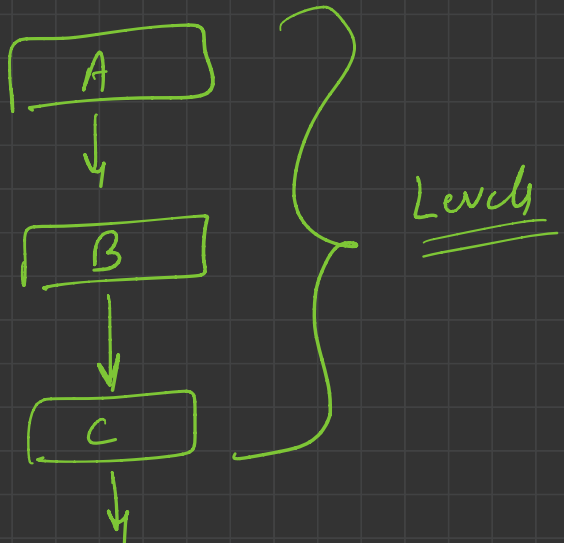
→ Single Inheritance

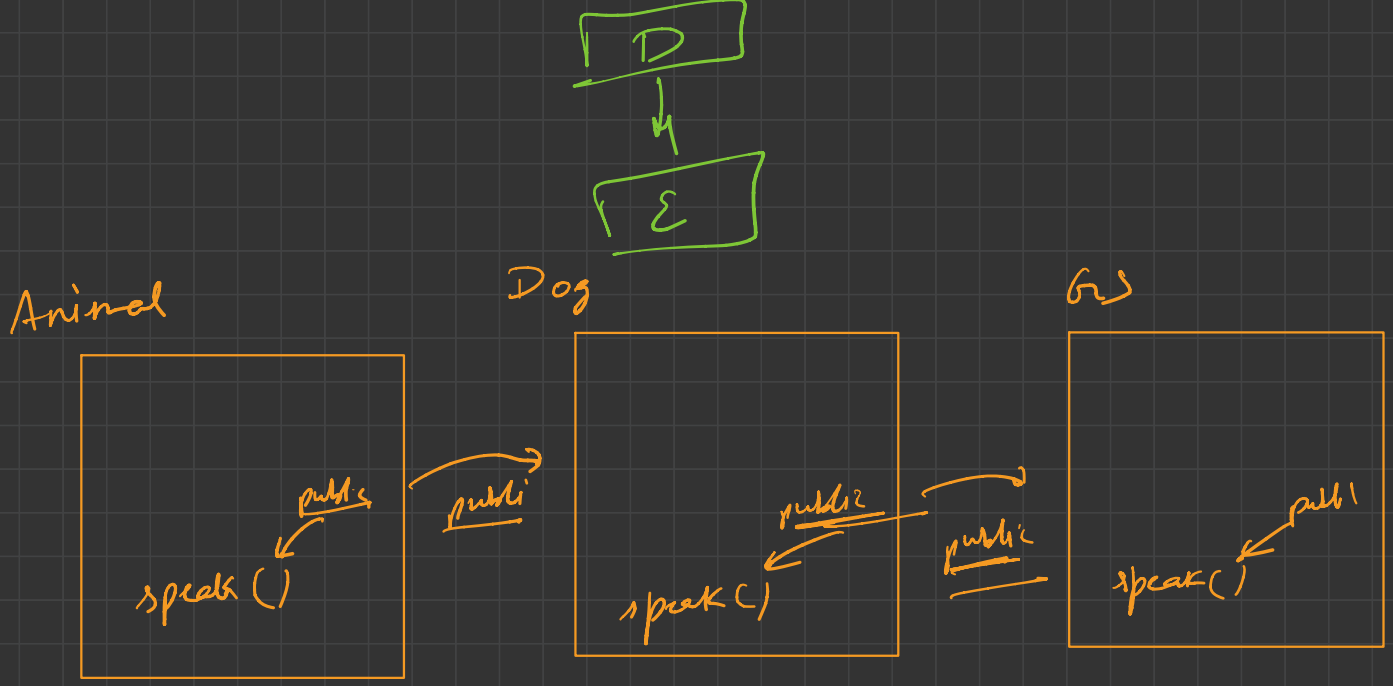




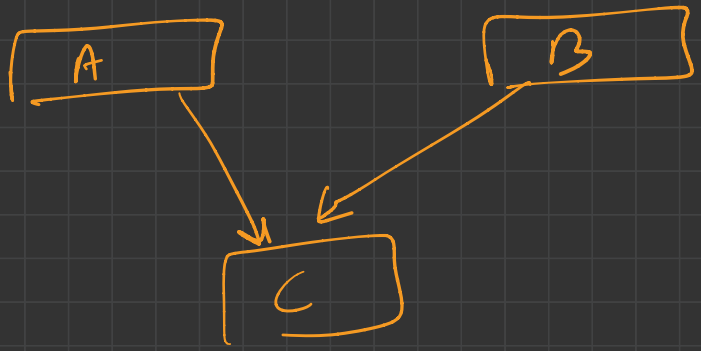
→ Multilevel -

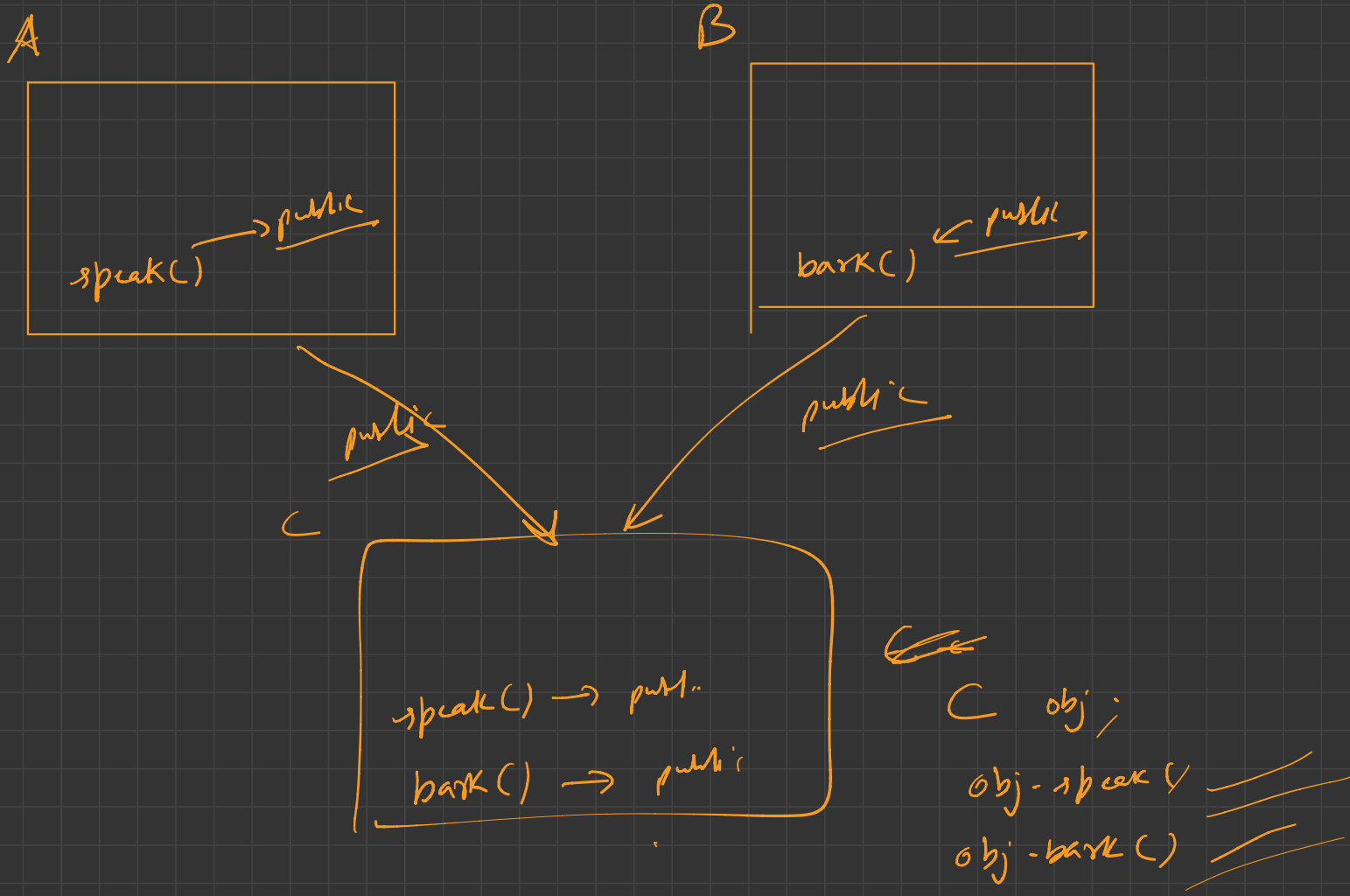





D
E
Animal
Dog
Cat
speak()
public
public
speak()
public
public
speak()
public

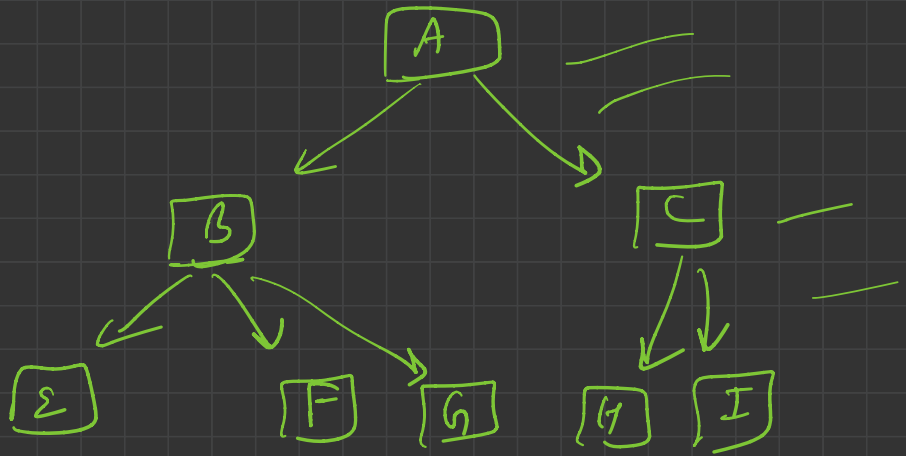
③ Multiple Inheritance

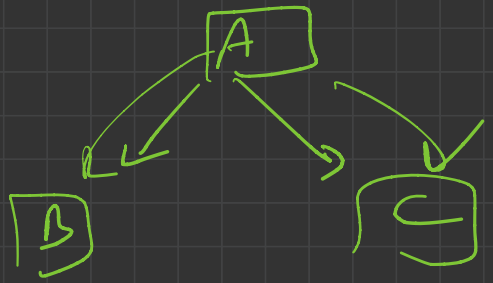



A
B
speak()
public
bark()
public
public
public
C
speak() → pub..
bark() → public
C obj;
obj.speak();
obj.bark()

(3) Hierarchical

↳ One class serve as Parent class for more than 1 class

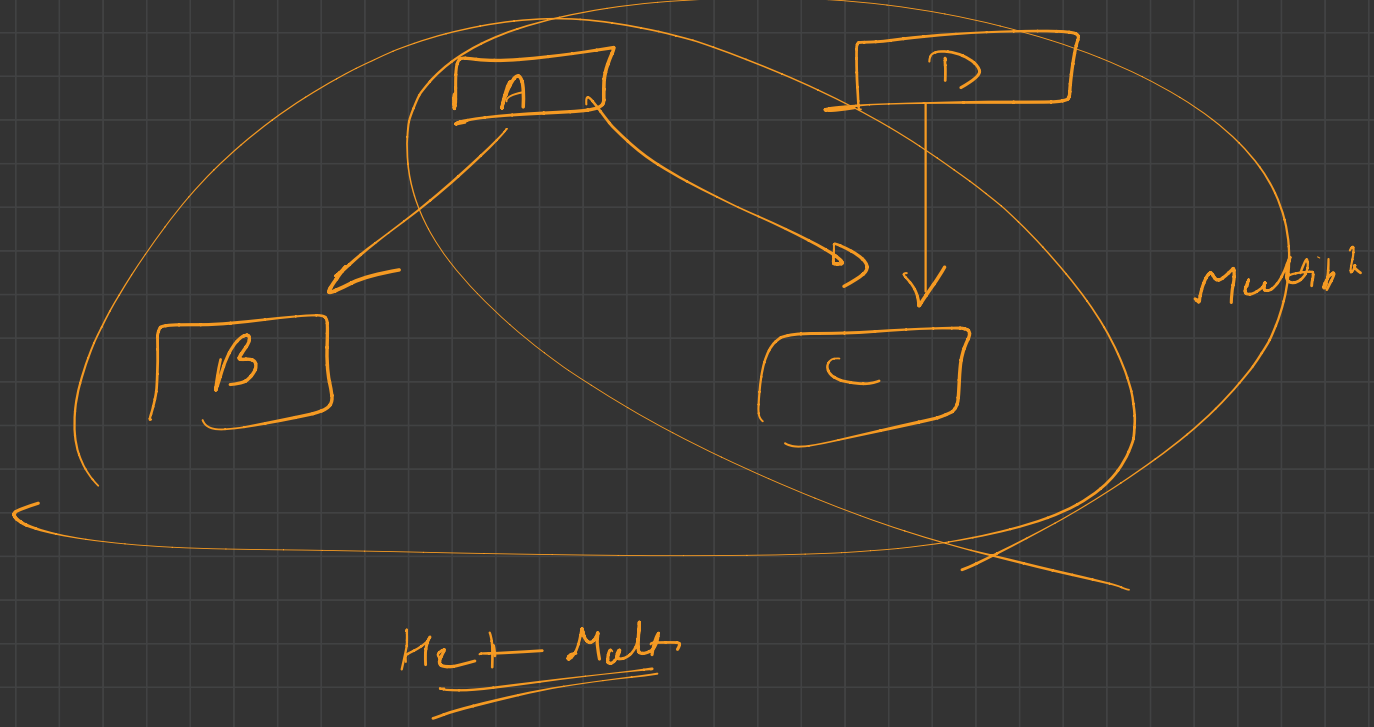




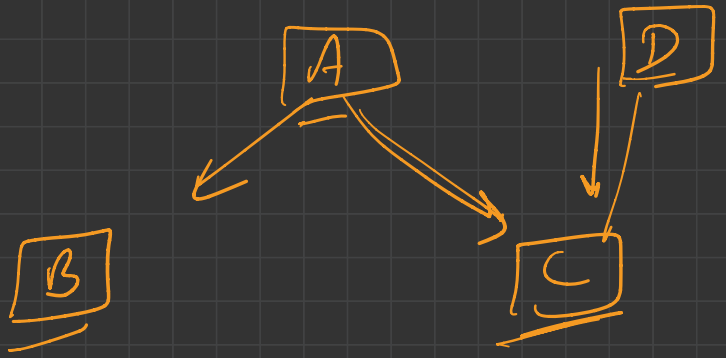
5 → Hybrid Inheritance

↳ Combination of more than 1 type of inheritance

Style / Nisa



Het Multi

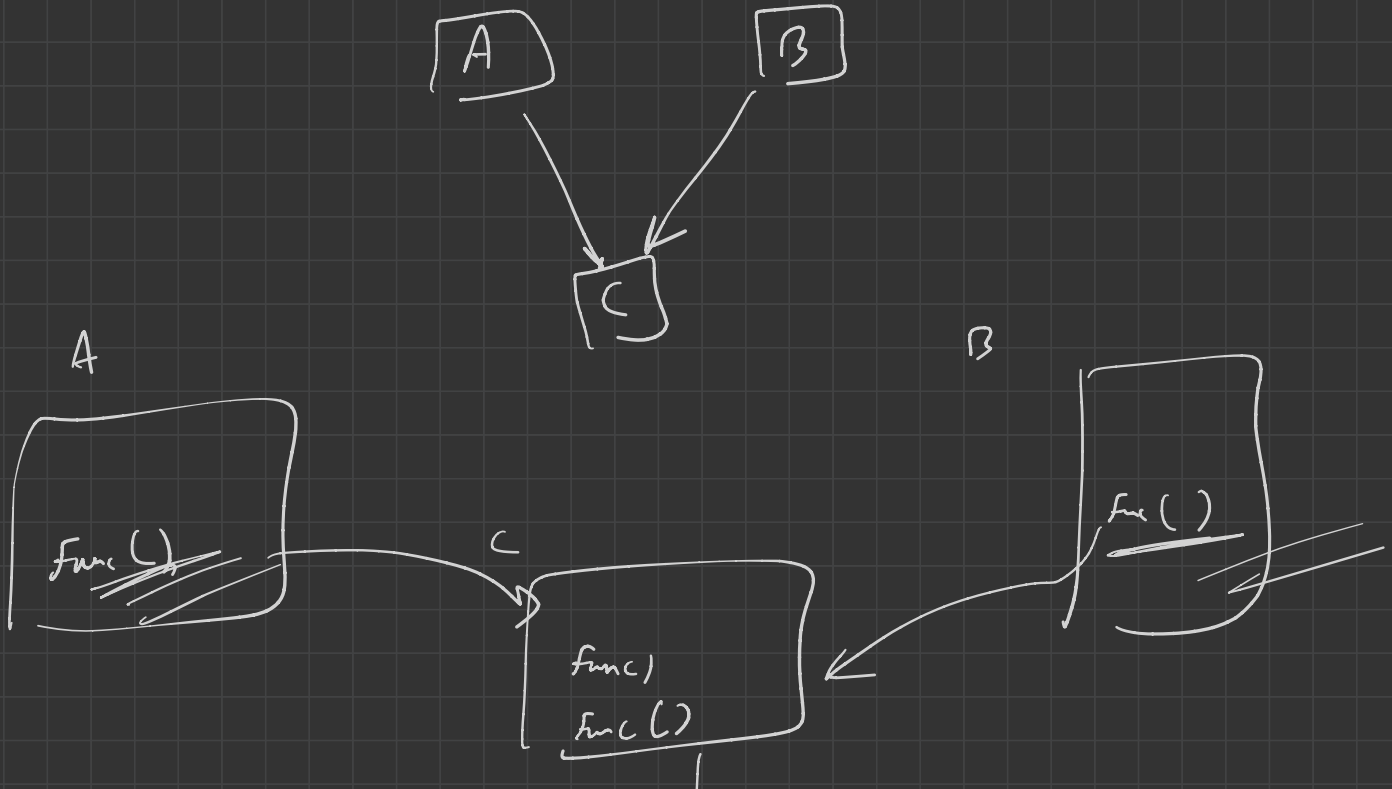


← Hybrid

```
class A
{
};
class D
{
};
class D : public A
{
};
class C : public A, public D {
};
```

H/w
↳ Explanation

→ Inheritance Ambiguity :-



↓

< obj

obj - Func() → ?

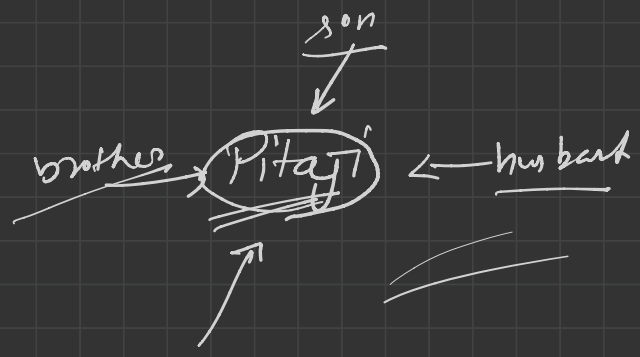
scope Res

op → ::

obj. A::Func();

obj. B:: Func();

→ Polymorphism :-



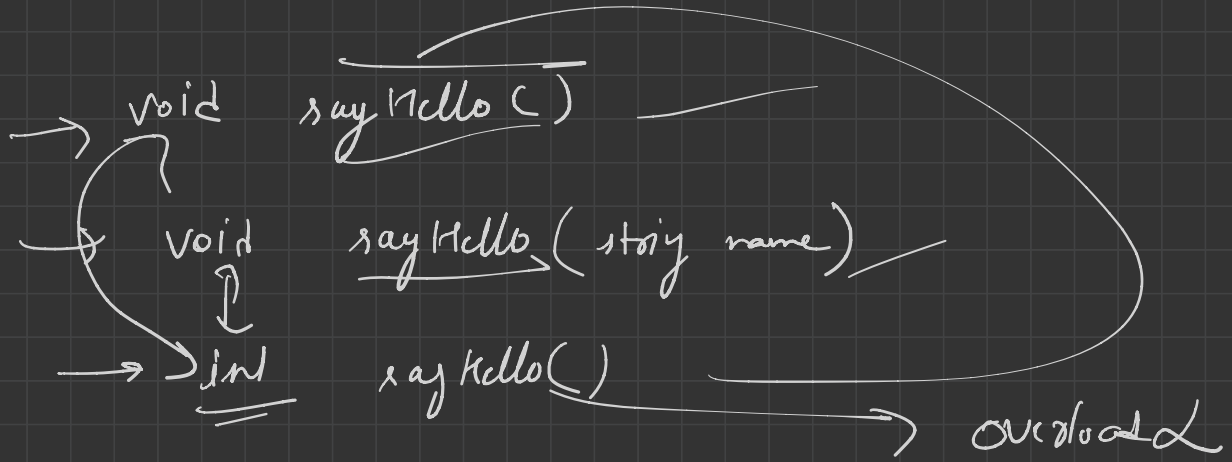
Polymorphism

↳ Compile time Polymorphism

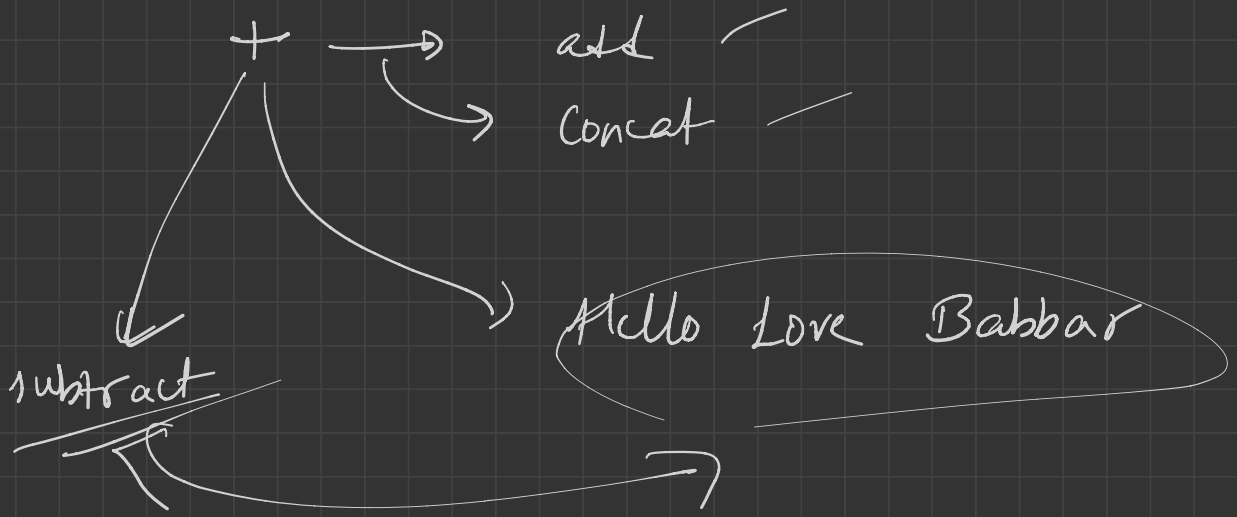
↳ Run - time Polymorphism

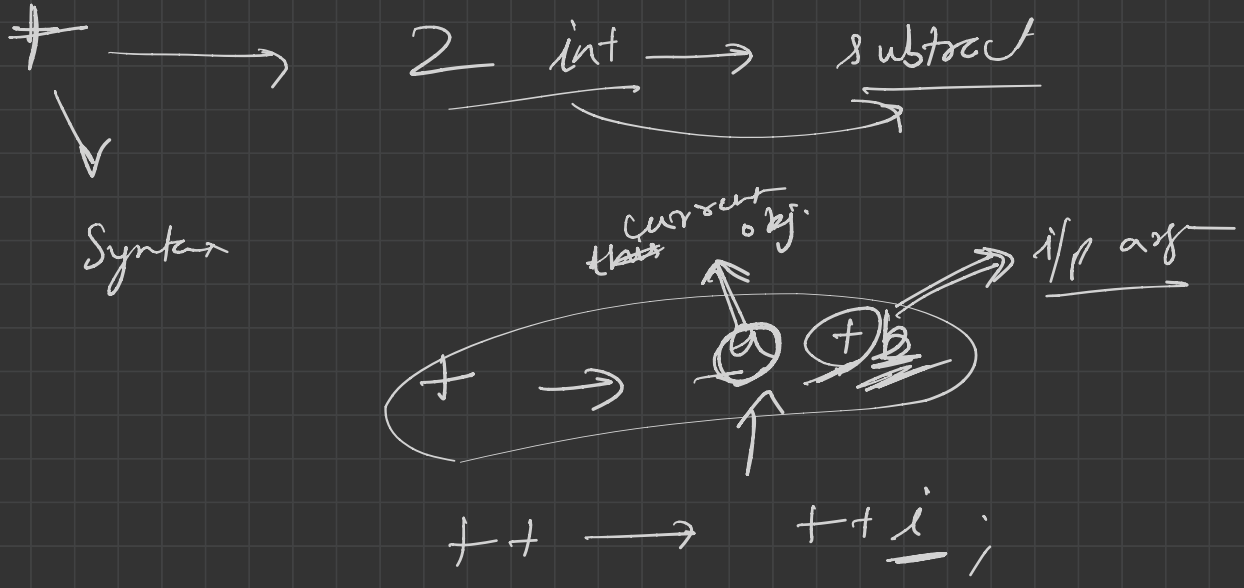
① Compile time Polymorz → Static polymorphism

- Function Overloading
- Operator Overloadig



→ Operator Overloading :-





return_type operator + ( ) ← i/p

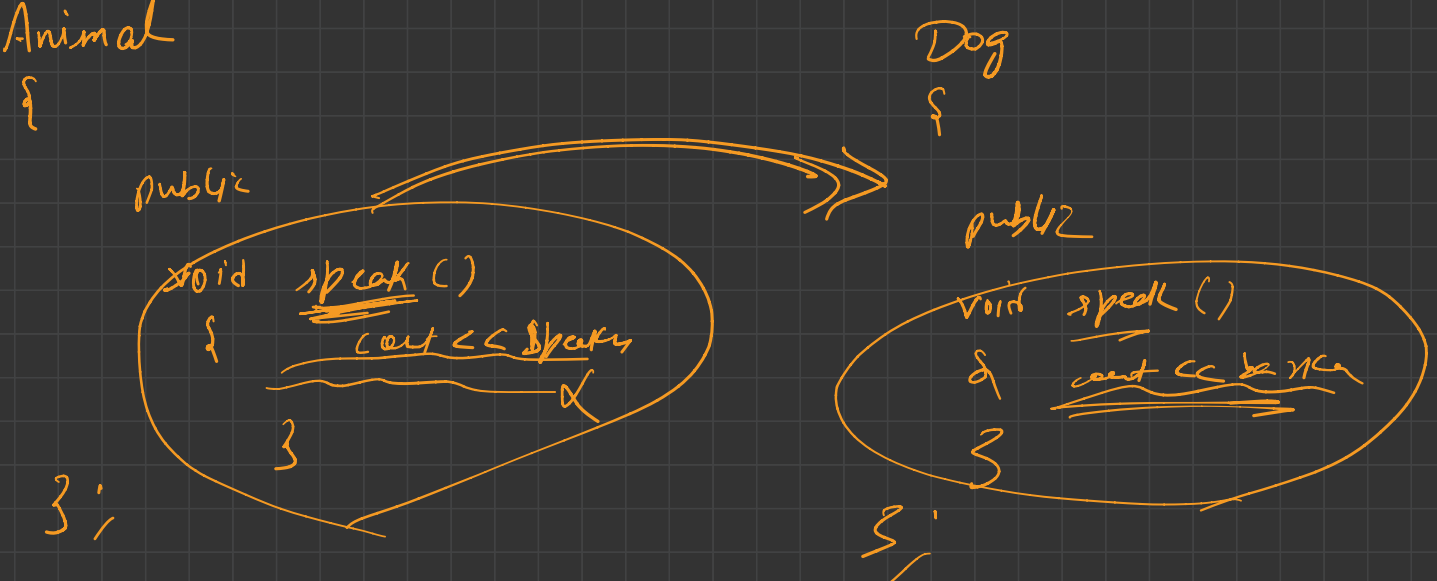
{

}

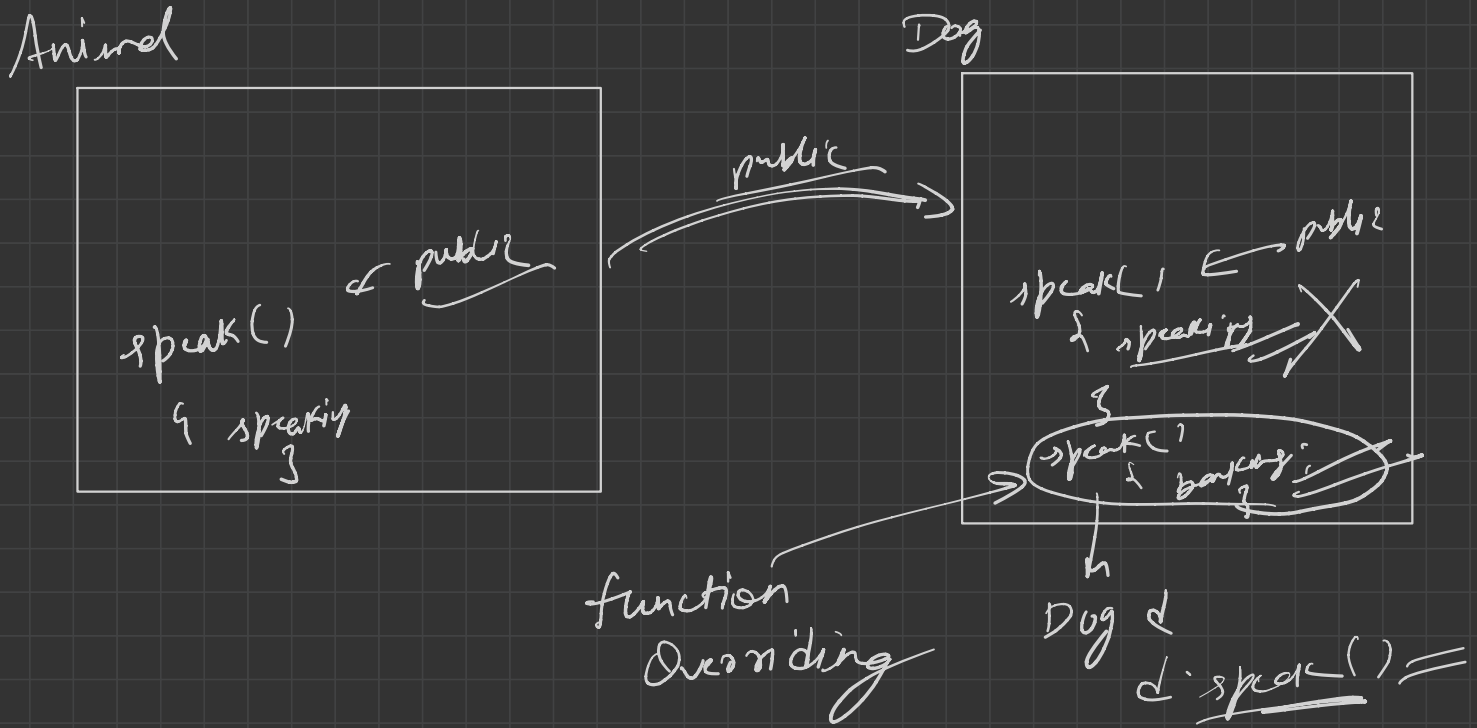
→ Run - time Polymorphism → dynamic polymorphism

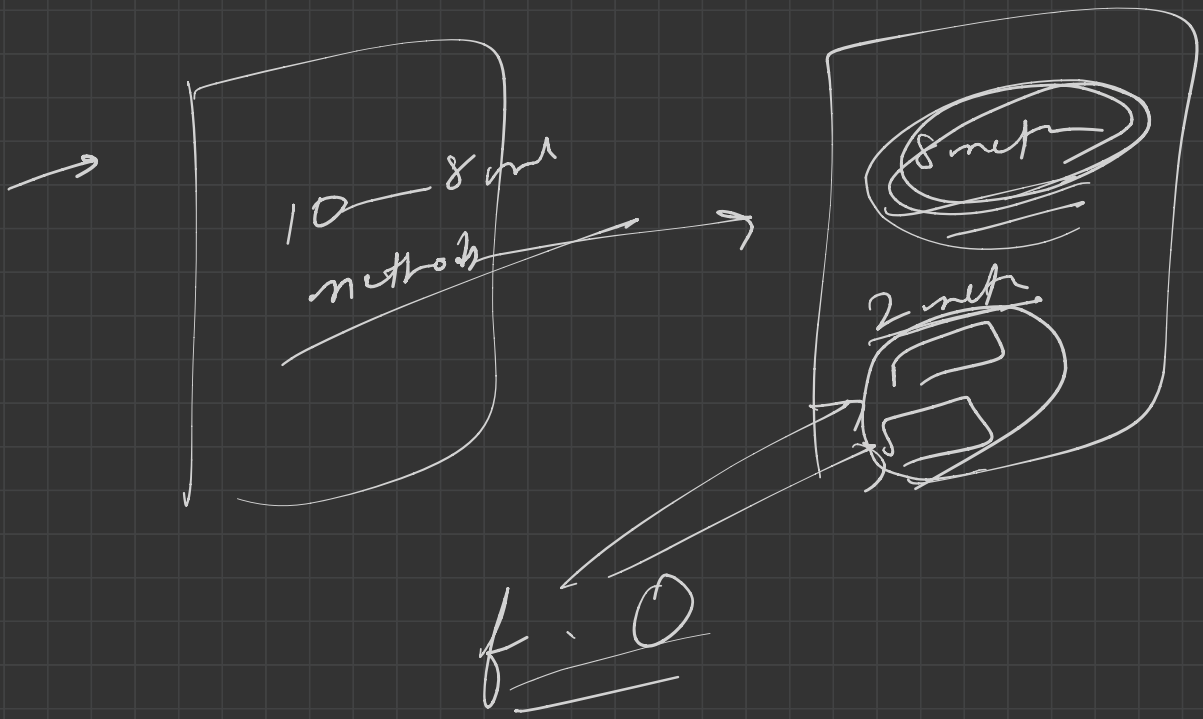
Inheritance dependent

↳ Method Overriding



→ Rules:- func name
func argu
inheritance





Encapsulation Inheritance Polymorphism

→ Abstraction :-

↳ "Implementation Hiding"

Advantage

Encapsulation / Abstraction

↳ S.O → Link Descr/bn