# OOPs - 3

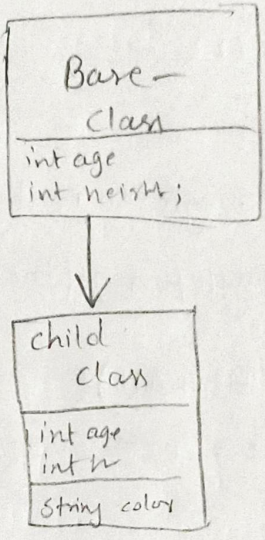
4 main Principles —

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

## Inheritence —

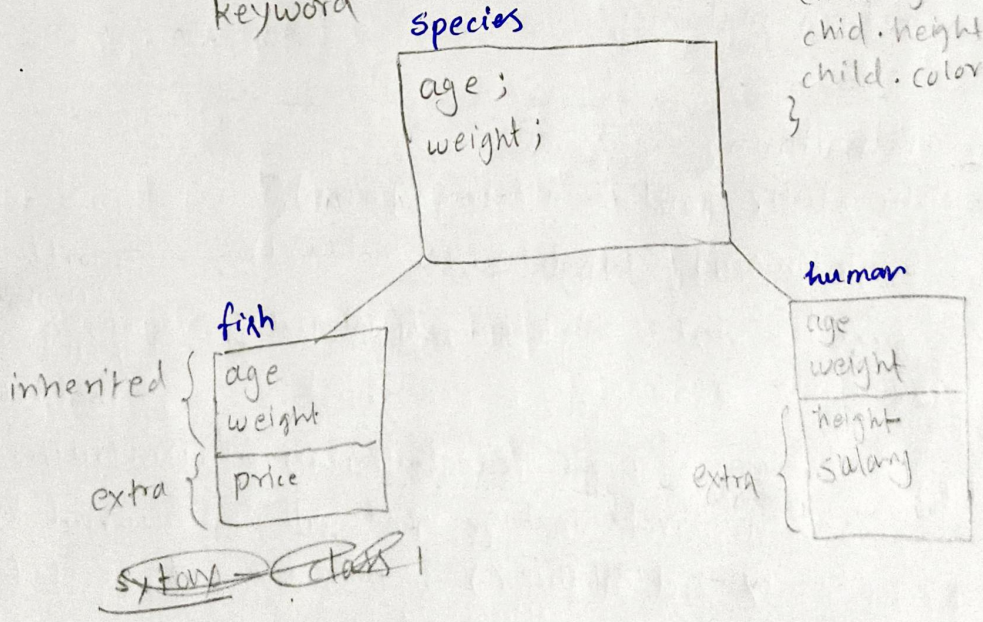
child class inherites properties from the Base class.

using "Extands" keyword

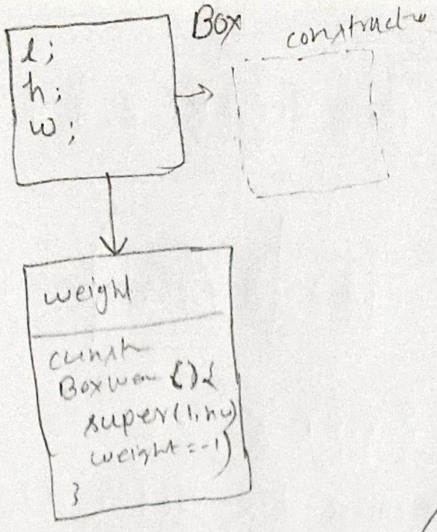



Syntax —

```
class Child Extands Base{
    int color
    ---
    ---
}
psvm( ){
child child = new child
child.age
chid.height
child.color
}
```




~~Sytax — class~~




public Boxweight (int l, int h, int w, int weight){

super(l, h, w);
this.weight = weight;

super(l, h, w) → this will call constructor of parent class.

# only public & protected values will be accessed in child class not the private.
for that we need to creat an method.

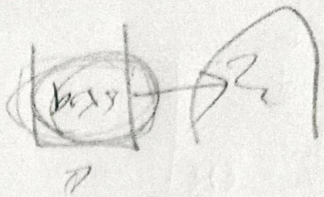
Dynamic Binding?

Ex- if we make 'l' in Base class private and try to access that using this.l or super.l we can't. But if have a function that can return 'l' and that method is public in base class, we can now access 'l' in child class.

& object of Base class only can access properties of it's own class not the child class's properties.

`Box box5 = new BoxWeight(2,3,4,10);`

When we try to access —



`box5.weight` → Error ?

Because the reference type is Box, & Box doesn't have field weight.

But if we try a method, —

it will work because methods are resolved at runtime.

Ex —

```
Box obj = new BoxWeight(2,3,4,10);
obj.message();
```

This will print "This is BoxWeight"

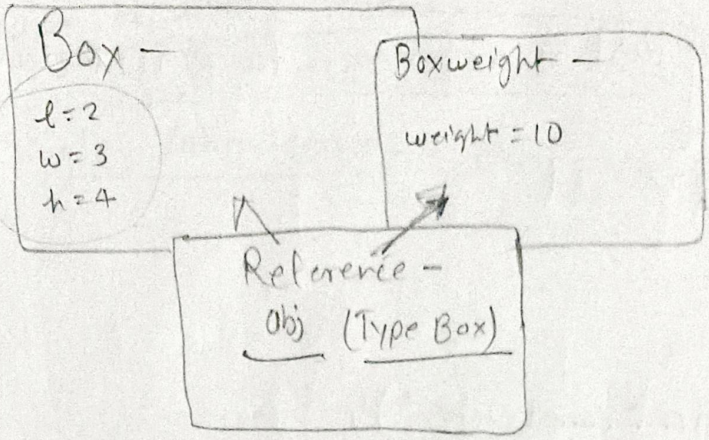
```
class Box {
    void message() {
        SOUT("This is Box");
    }
}
class BoxWeight extends Box {
    @override
    void message() {
        SOUT("This is BoxWeight");
    }
}
```

why

At compile time → `Box obj` → check for method message() →

At runtime → `BoxWeight obj` → & runs BoxWeight's message();

Dynamic Binding -




So, obj has type box But pointing to class "Boxweight" that's why we're not accessing obj.weight because – Reference type ~~decide~~ decides which field/method are accessible.

---

What about –

```
Boxweight box = new Box(2, 3, 4, 10);
```

& try to access weight variable –

→ So basically, the reference variable box has type Boxweight & that's why we can access the weight –

**But –**

The very first line will give error As the no of passed arguments are 4 but we are calling constructor of Box which only take 3 arguments & that's why the 4th argument will cause error & that's why weight will not be initialize & can't be access.

Read b/w the lines....

super() ? → just a reference to parent class

That's all

```
class A{
   A(){
      SOUT("Parent constructor");
   }
}
class B extends A{
      B(){
        //SOUT X → we need the super first
        super(); ——→ implicit
        SOUT("child constructor");
      }
}
```

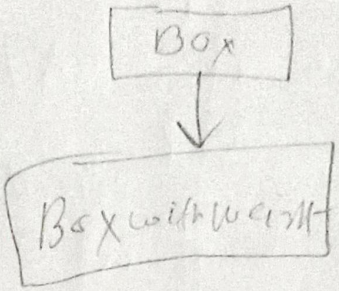
But if A doesn't have a no-arg constr
we need to call (required) the super in B

# can't use super in static contex,
it req an obj.

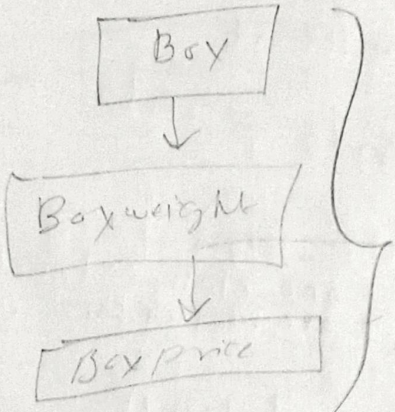
# Always point to immidiat parent

# Types of Inheritence -

(1) Single Inheritence - one class extends another class -




(2) Multilevel Inheritence - #




Basically we have a code -

```
Boxweight(Boxweight other){
    super(other);
    this.weight = other.
}
```

Next page

What happende? -

there is constructor in Box which take orgument of type Boxweight -

But, java allows you to pass a child object refere wherever a parent type parameter is expected.

```
Boxweight b1 = new Boxweight(1,2,3,10);
Boxweight b2 = new Boxweight(b1)
```

in above constructor - we pass b1 to super & here we assign this.l = b1.l → present in b1
this.w = b1.w →
this.h = b1.h →

## Features of multilevel —

→ Transitive property —

if C → B

B → A

then indirectly —

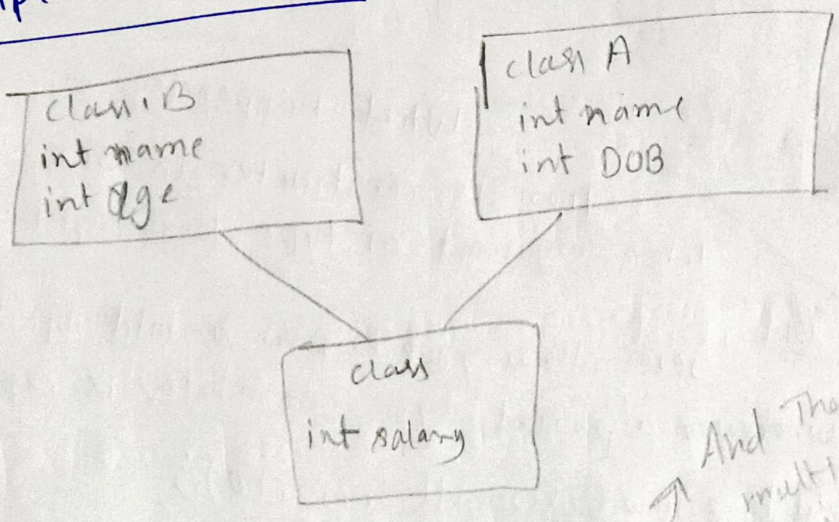
C → A

```
A {
  int x;
}
B extend A {
  int y
}
C extend B {
  int z
}
```

→ only public & protected methods got inherit

→ child can override parent method —

## Multiple Inheritence — one class extending more than one classes —

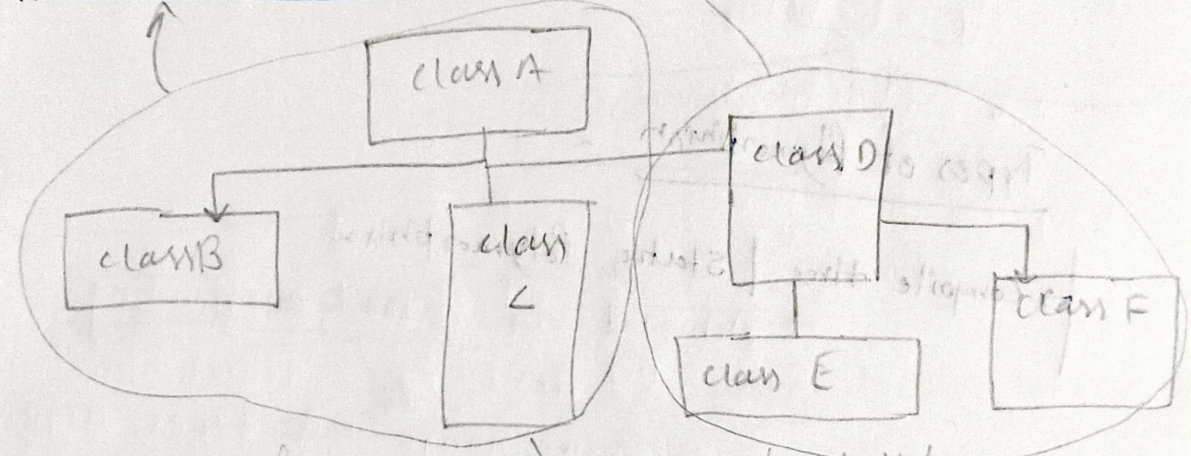


But what if we need that, we need to have a child multiple parents ?

→ Then we use **Interface** ↑ later in this course we'll see what are Interface but for now —

They are class like but they don't deal with values only provides templete for child class.

**Hierarchical Inheritance** →



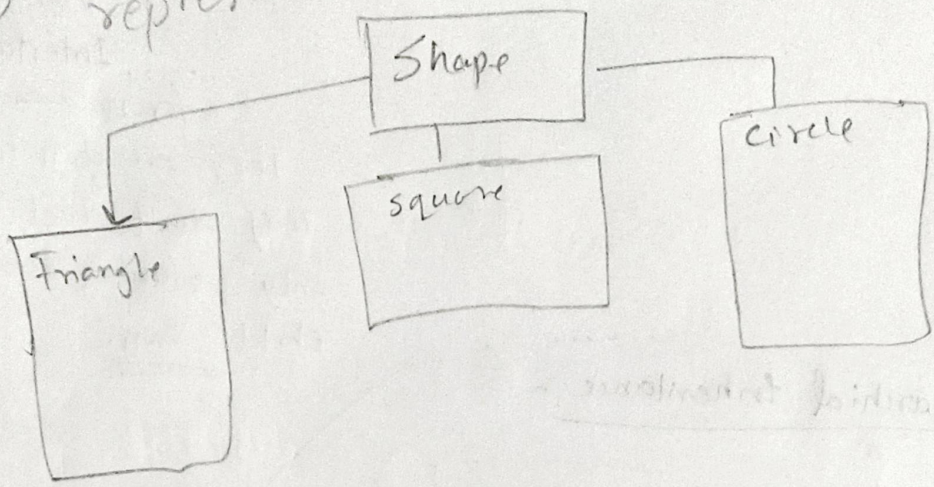

Same rules for single level & multiple level rules

Hybrid Inheritance

Not in Java

# Polymorphism -

many → ways to represent




## Types of Polymorphism -

↳ Compile time / Static Polymorphism

→ Achieved via method overloading

same name but types, argument & return type & ordering can be diff —

Ex — multiple constructors.

↳ Runtime / Dynamic Polymorphism -

→ achieved via method overriding —

only body will be diff otherwise returntype, arguments etc will be same as in parent class.

Overload → same class

Override → Two diff class extend one another

---

`Parent obj = new Child()`

which method will be called ?

depends of child.

⇓

Known as **upcasting**

---

final → to prevent override ,

& also a class from being inherited

⇒

# can we override the static ?

⇓

No

⇒ we can inherit but cannot override.

## Encapsulation — This is basically — wrapping up data & code together into a single unit

hides the internal details but exposes controlled access

Achieved by
- → private fields
- → public fields

Why we need this -? i) data hiding
ii) control
iii) flexibility & reusability

---

usually we use private fields & use getter - setter methods to access & modify data —

⇒ Diff b/w Encapsulation & Abstraction ?

## Abstraction – Hiding implementation details & showing only the essential features.

& basically it works mak like user's need –

& what is user's need?

→ what it does? not How?

Ex – As Always (Cars)

{ we only need to know Break, accelerator & steering to drive it not the how engine, fuel injection works.

This is abstraction

## How To Achieve

→ Abstract class
↳ (0 – 100%)

→ Interface
↳ 100% before Java 8
now we can have default/static methods

## Diff –

| Encapsulation | Abstraction |
|---|---|
| → Internal ~~implementation~~ data hidden | implementation hidden |
| → focus to protect data | focus to design system |
| → [private, public] } Achieved By | → Abstract class, Interfaces |