→ Inheritance

: this & super

→ Polymorphism

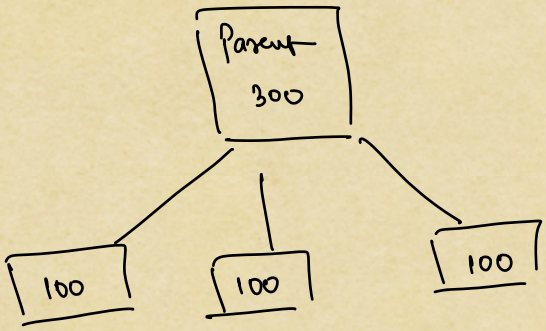
→ Static [time]

→ Assignment → (2)

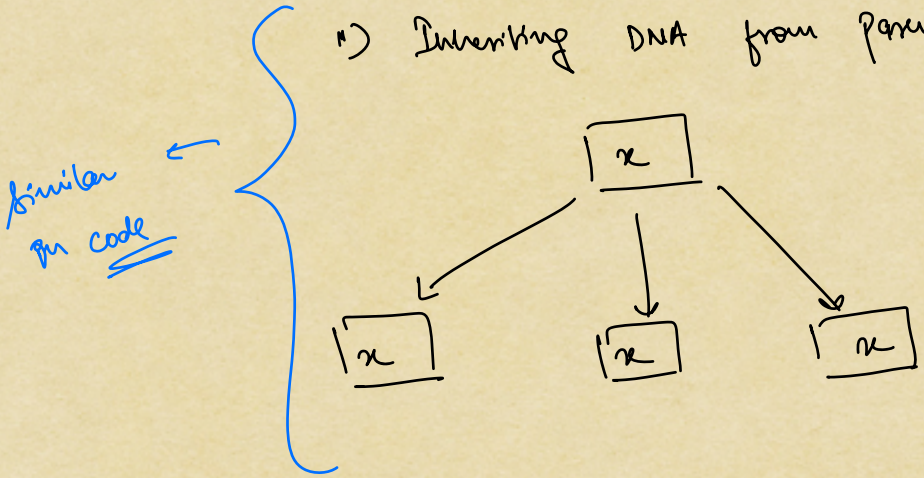
⇒ Inheritance :-

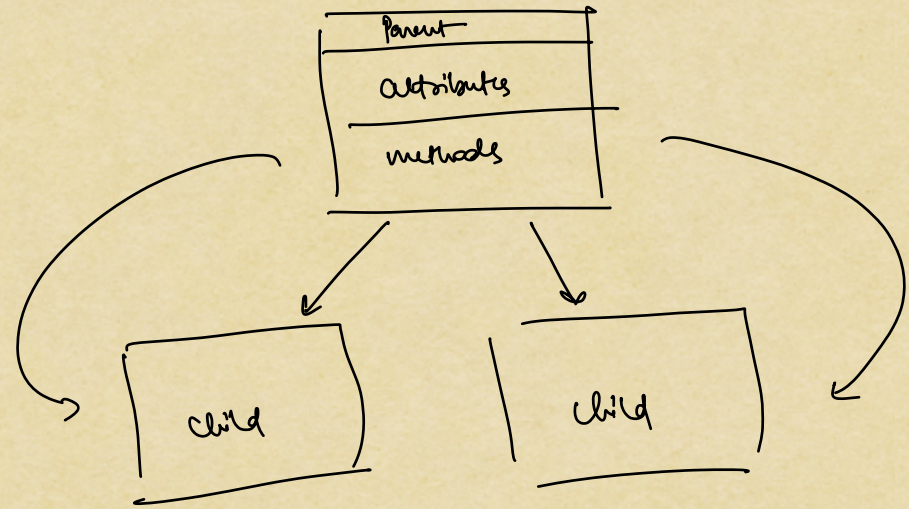
* Inheritance in real life :

1) Inheriting property / assets / money :-



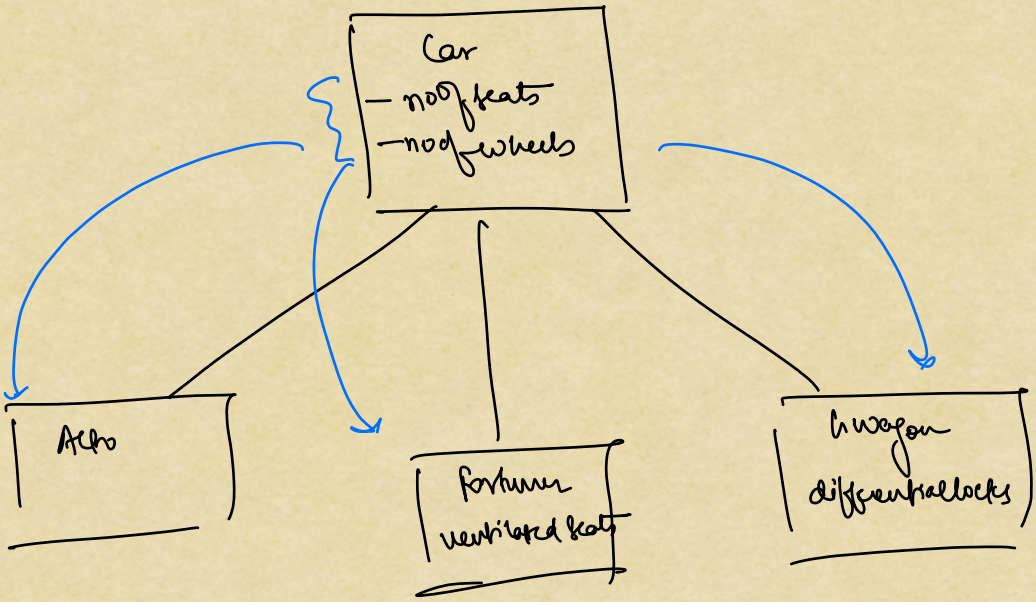
2) Inheriting DNA from parents





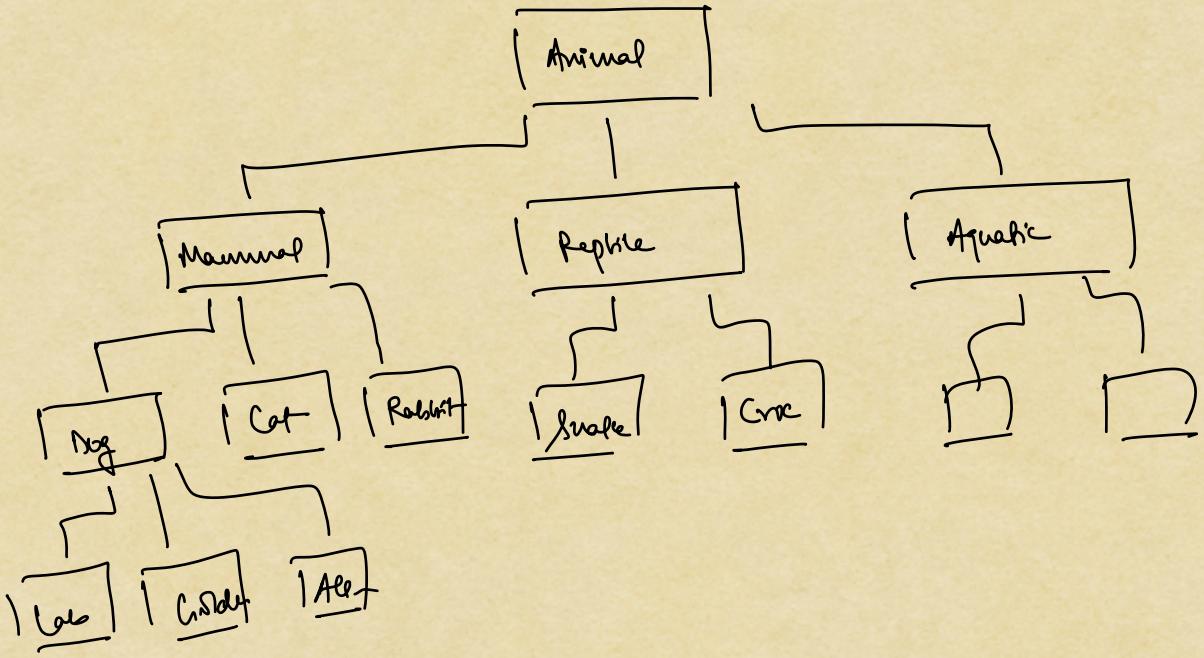
→ Why do we have inheritance?

- → Reusability
- → Reduction in duplicate code
- → maintainability becomes easier [approx...]
- → Extensibility



→ Maintaining hierarchy / grouping things together

→ is-A relationship



Labs is a dog

Labs is a mammal

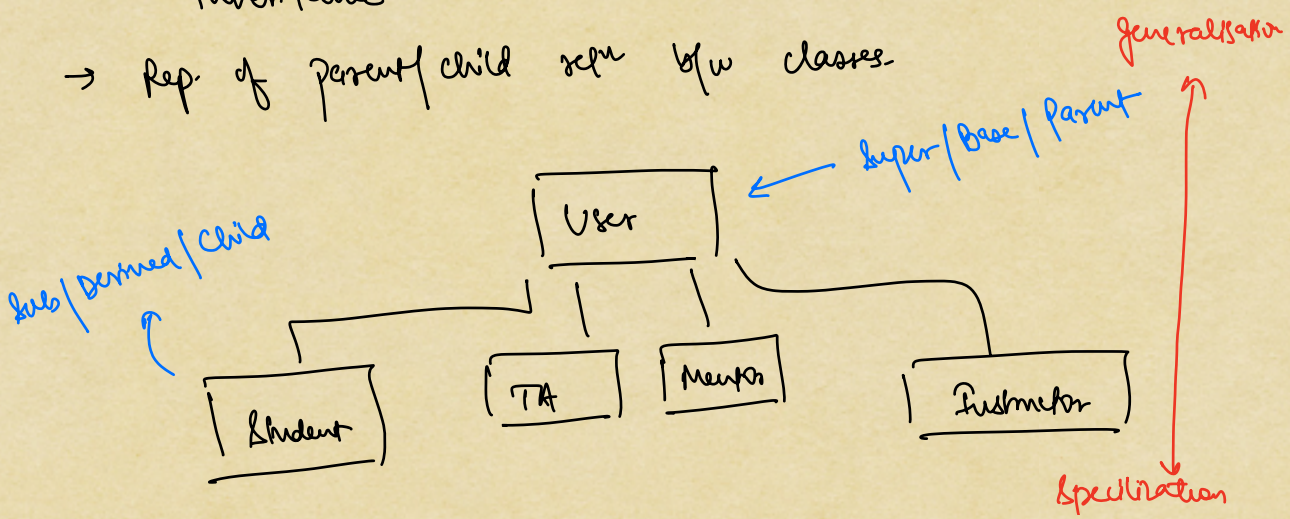
Labs is an animal

Dog is a mammal

Dog is an animal

→ Representation of hierarchy in classes is known as inheritance

→ Rep. of parent/child rel^n b/w classes.

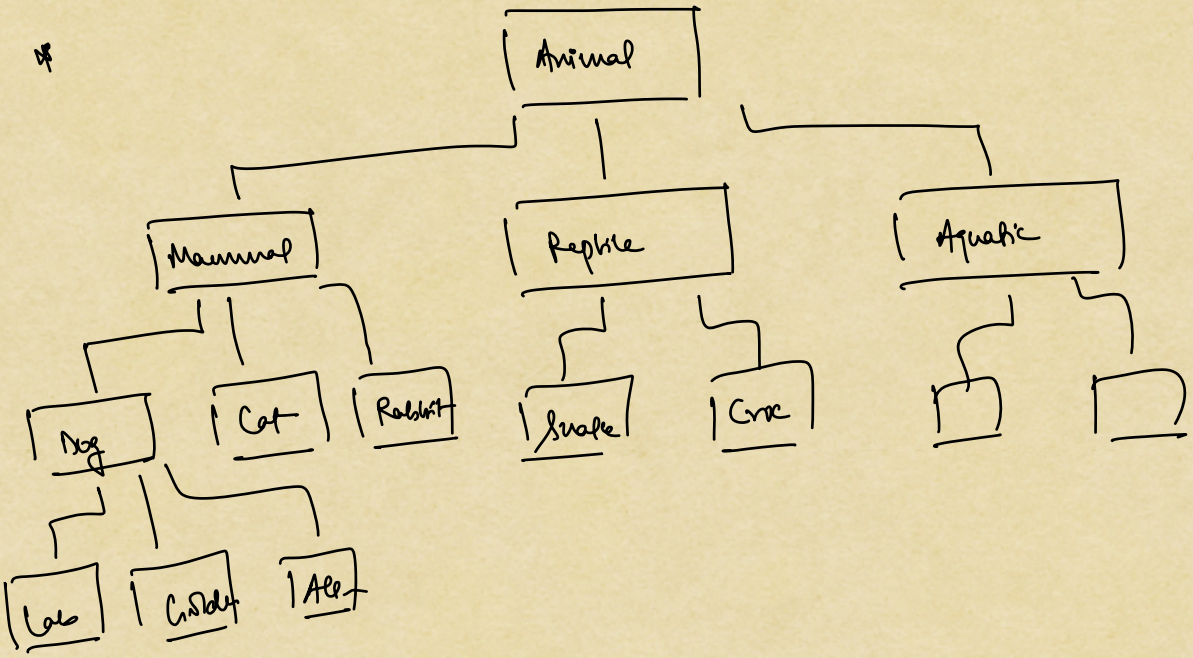


* child classes inherit all members (attributes + methods) of parent but may or may not have their own members

all => private as well

* child class will have all attributes of parent class but accessibility depends upon access modifier

*

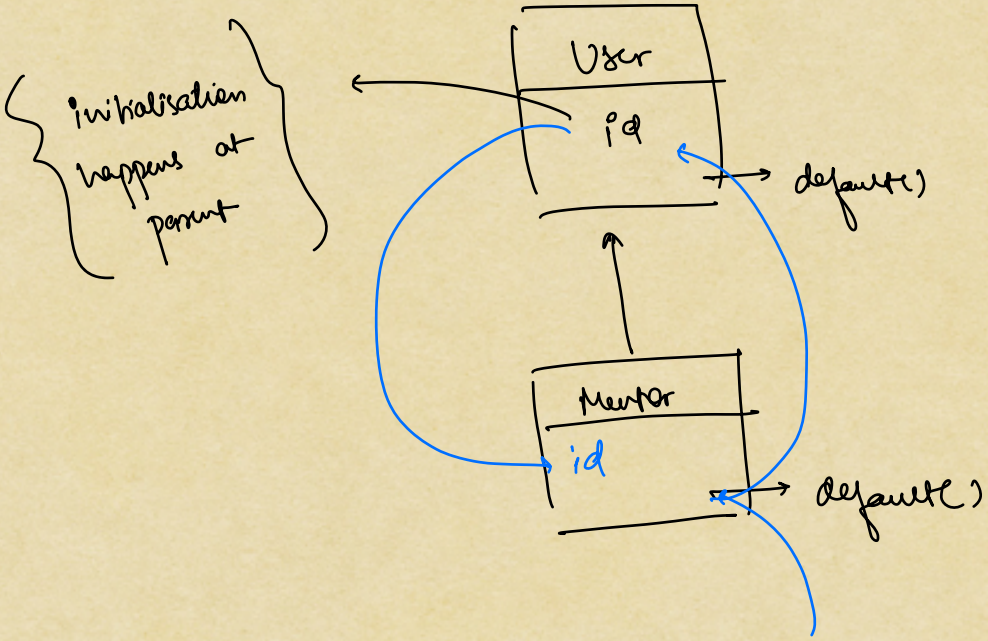


→ We get the idea of a class just by looking at the hierarchy and no need to look into implementation

→ Abstraction

→ STEPS TO CREATE OBJECT OF A CHILD:-

* assume there is no constructor present



Mentor m = new Mentor();

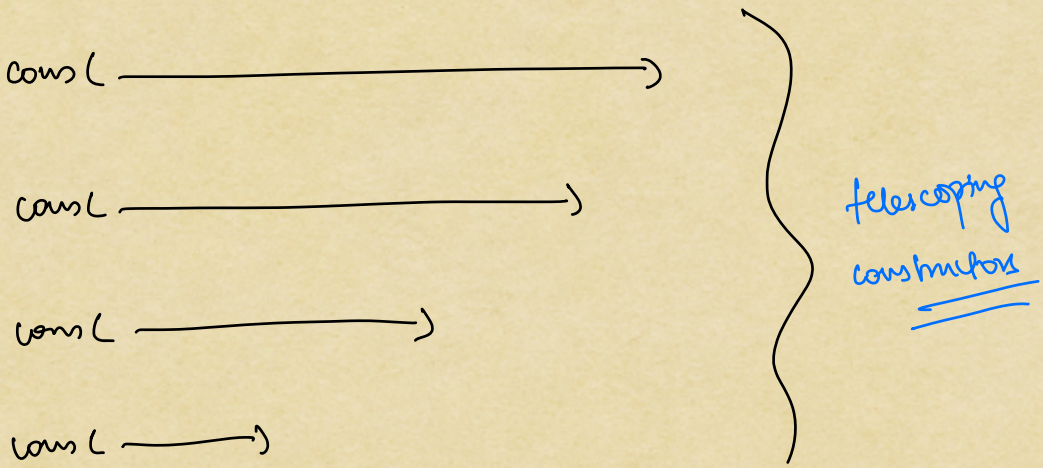
Mentor() {

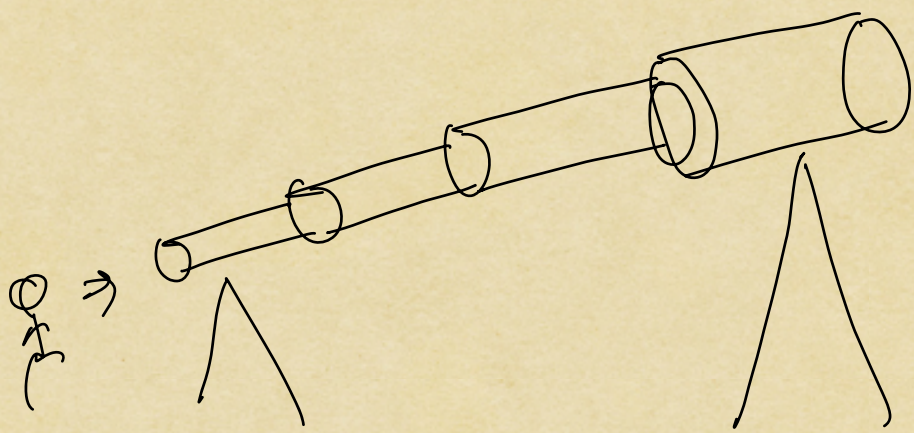
→ Call to User(); { super }

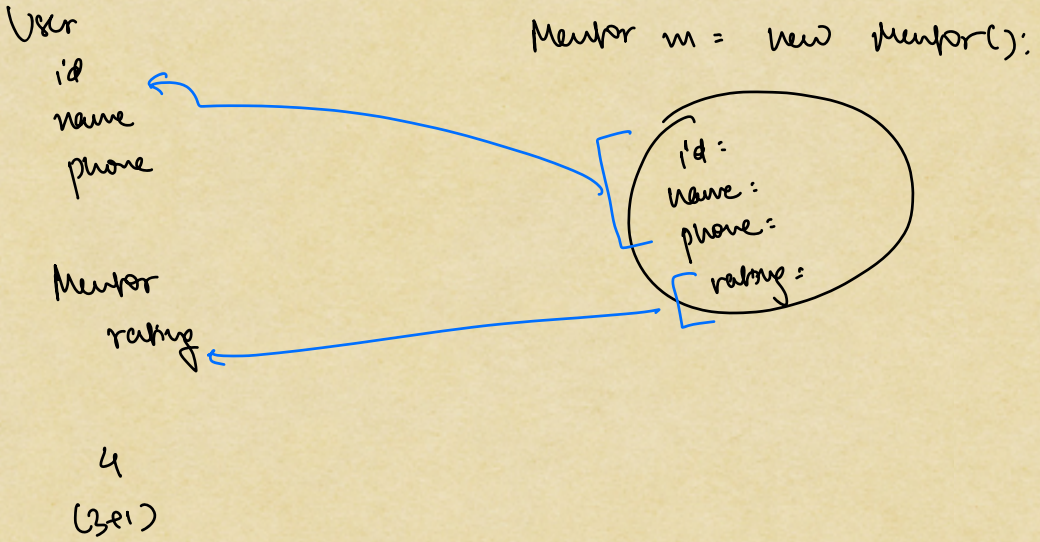
}

⁘ this & super:-

	this	super
Constructor	current class obj	parent class obj
attribute	current class obj	parent class obj
method	current class obj	parent class obj


cons(
cons(
cons(
cons(
telescoping
constructor




User
id
name
phone
Mentor
rating
Mentor m = new Mentor();
id:
name:
phone:
rating:

4
(3e1)

1

Mentor → default →

```
public Mentor() {
    super();
}
```

User ⇒ dont have any constructor → default

2

Mentor → has a cons →

```
public Mentor(int rating) {
    super();
    this.rating = rating;
}
```

User → no cons [default is present]

3

Mentor → has no cons [default cons] →

```
public Mentor(id, name, phone) {
    super(id, name, phone);
}
```

User → 1 param constructor [no default constructor]

User(id, name, rating).

IV

Mentor → 1 param

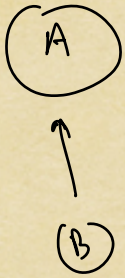
User → 1 param

```
public Mentor(int rating) {
    this.rating = rating
}
```

```
public Mentor(id, name, phone, int rating) {
    super(id, name, phone);
    this.rating = rating;
}
```

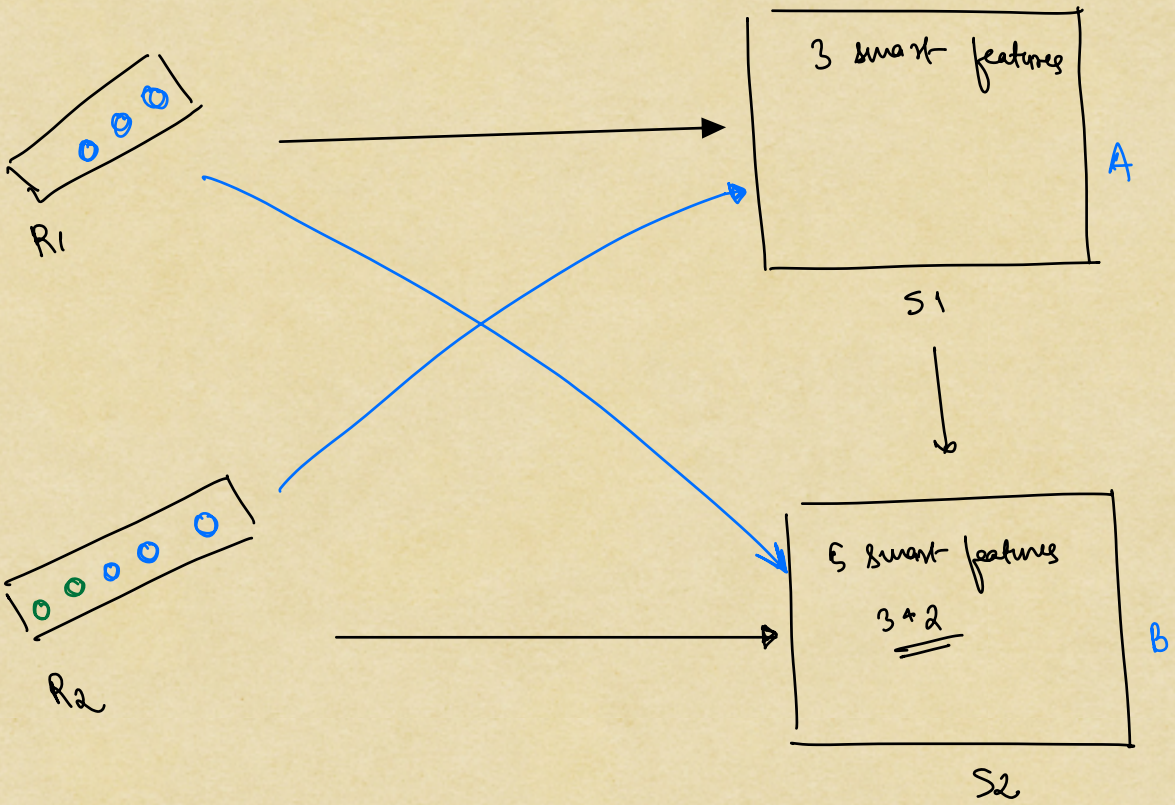
* If your parent class doesn't have a default constructor, then we will need to pass args for current class via child class cons. using super keyword

⇒ POLYMORPHISM



✓ `A a = new A();`
✓ `B b = new B();`

✓ `A a = new B();`
✓ `B b = new A();`



$R_1 - S_1$ ⇒ `A a = new A();`

$R_2 - S_2$ ⇒ `B b = new B();`

→ $R_1 - S_2$ ⇒ `A a = new B();` [upcasting] — implicit casting

⇒ doable ⇒ Yes [wrik code]

⇒ all the features in the class wont be accessible

⇒ not prone to errors

→ R2 - S1 ⇒ B b = new A(); [explicit casting / downcasting]

⇒ double = Yes [work code] | Runtime exception

⇒ all the features of the child class wont be accessible

⇒ prone to errors ⇒ highly

→ Method overloading & Method overriding

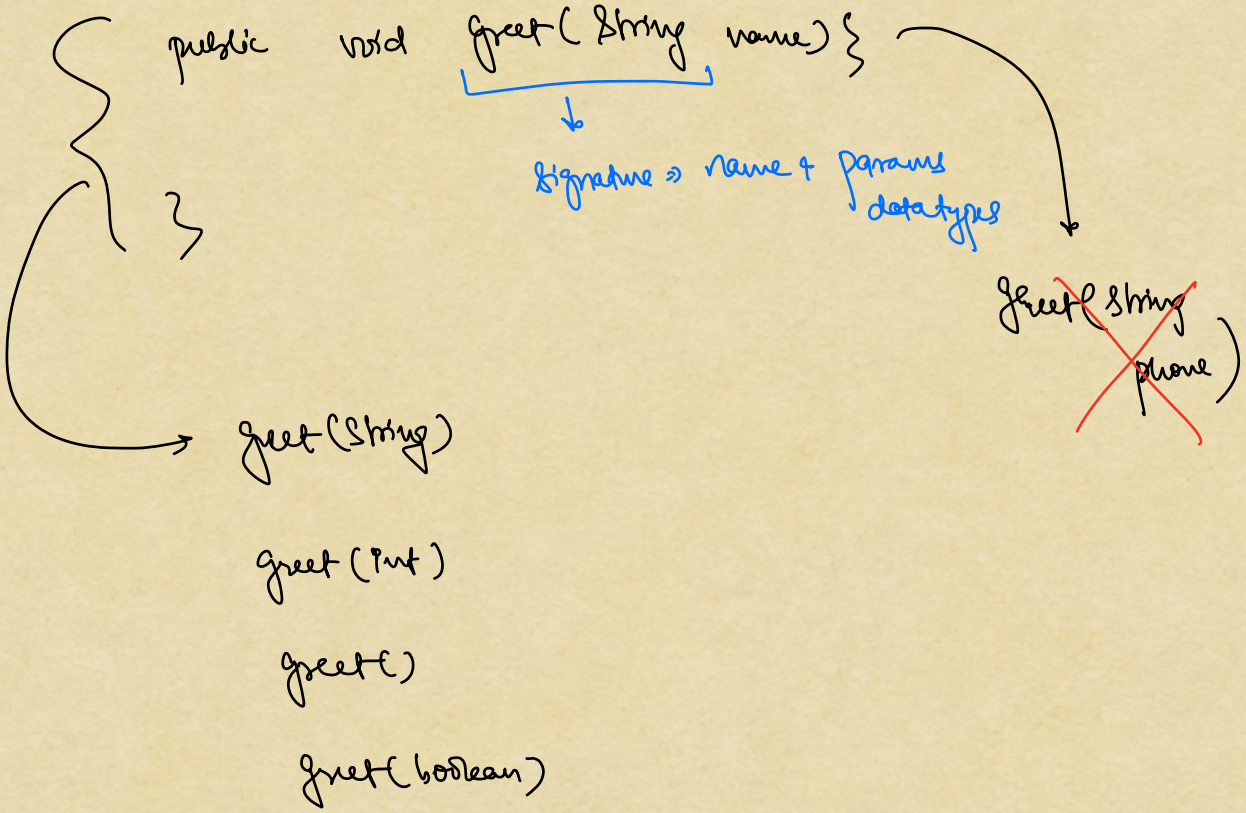
⇒ Method Overloading

```
public void greet(String name) {
    sout("Hi" + name);
}

greet();
greet(1)
greet(true);
```

not possible

⇒ Java identifies methods using method signature.



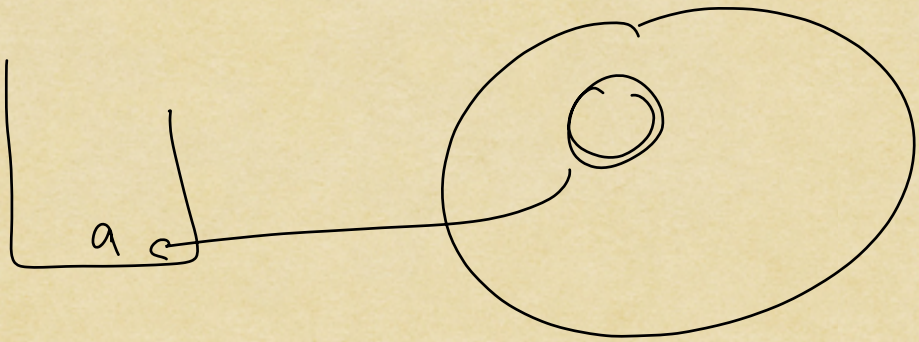
⇒ Can we have duplicate method signatures in a class, ans : No.

⇒ Can we have duplicate method names & diff params class, ans : Yes.

↳ Method overloading

Method overridding

A a = new A();

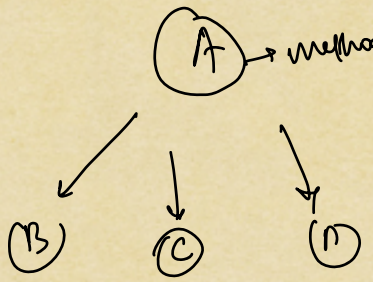


A a = null;

Use case ⇒ upcasting & this/super

↓

overridding



A a = new B();

a.methodA();