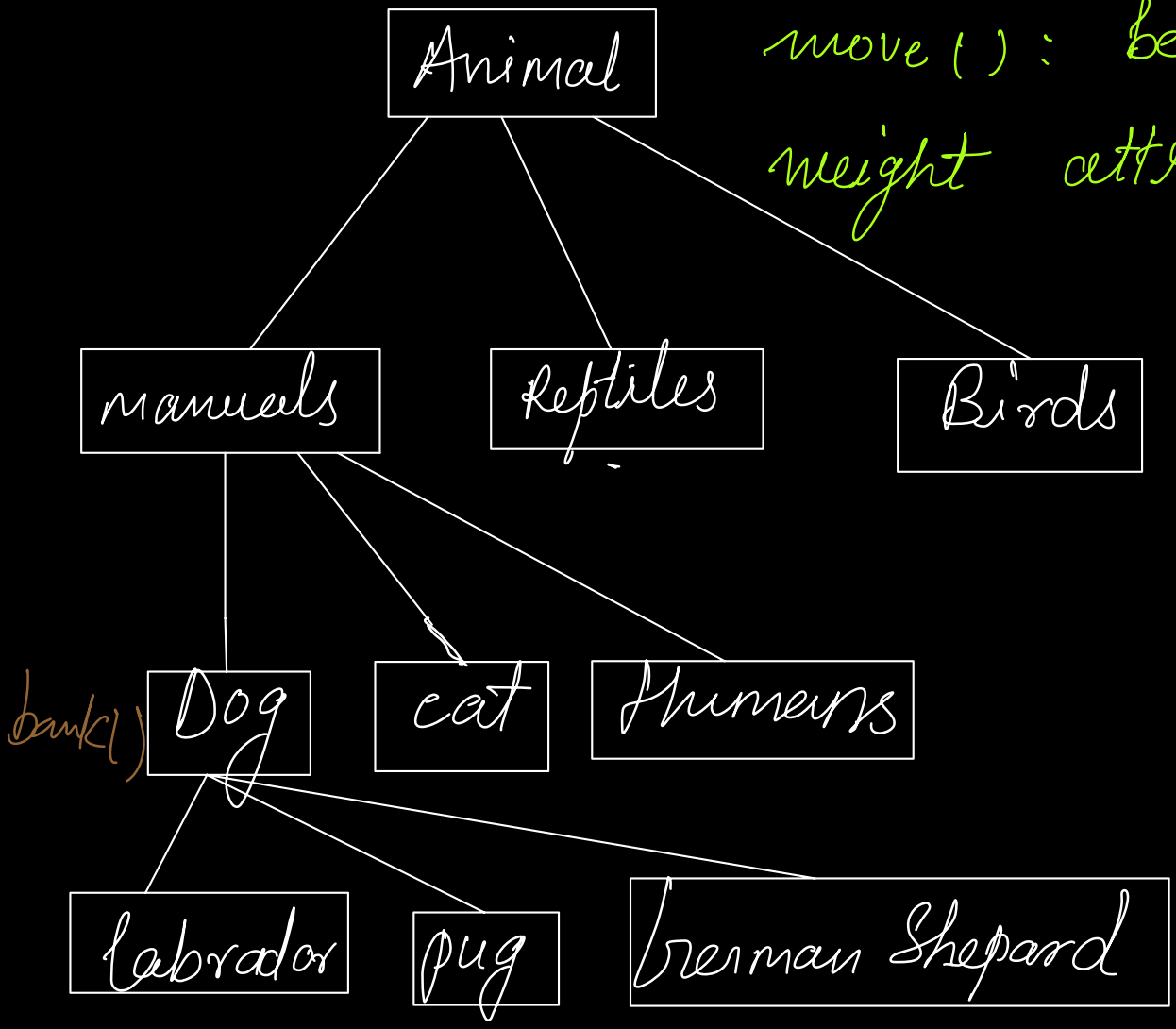
Today Agenda :-

Inheritance
Constructor chaining
Polymorphism
- method overloading (C)
- method overriding (D)

Inheritance :-

OOP is based on thinking about s/w system similar to how humans think about real life.

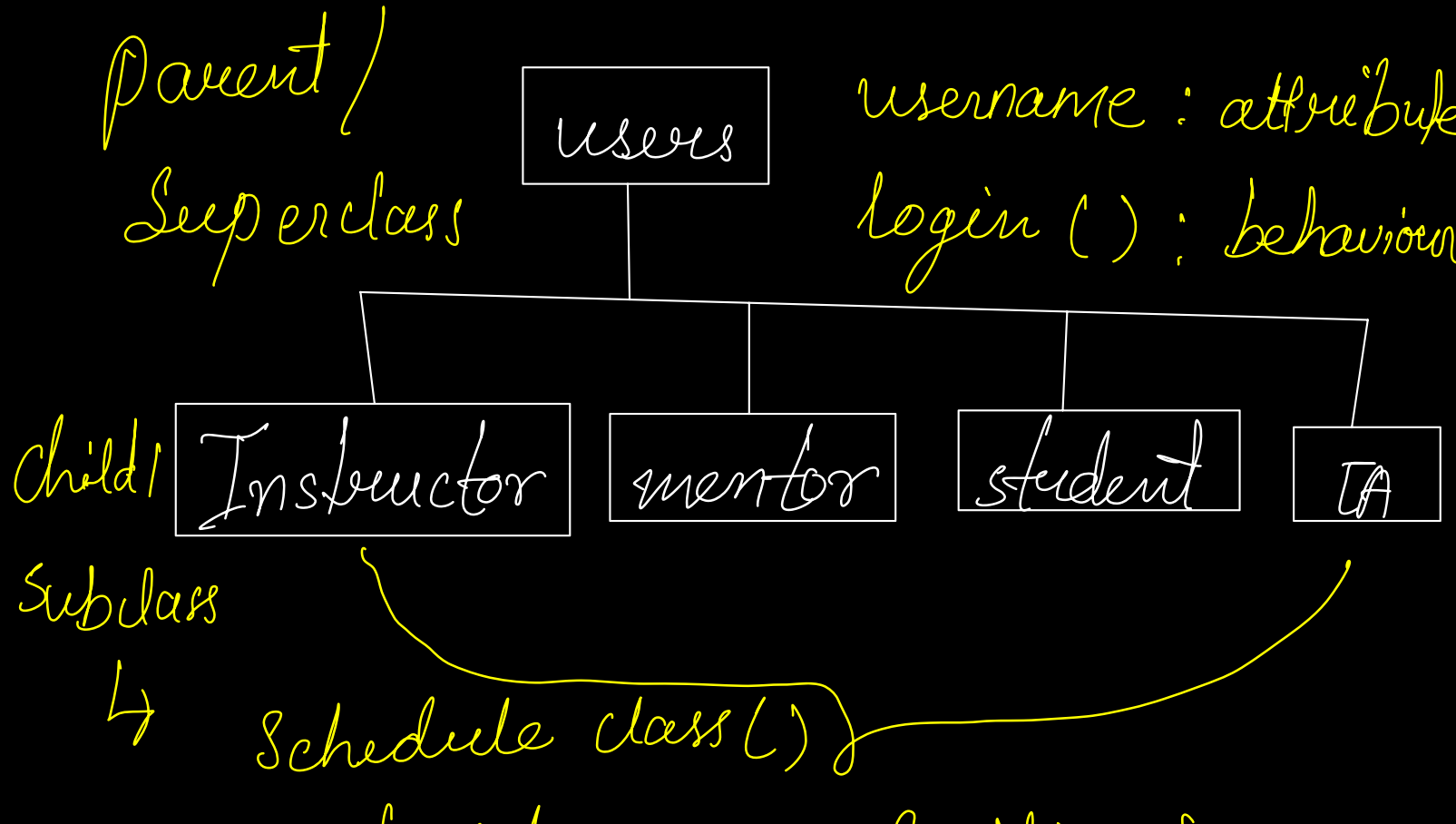


Heirarchy of representation allows you to share attribute & behaviour among specific categories.

Representation of Heirarchy among class / entities

↓

Inheritance } represented as parent - child relationship

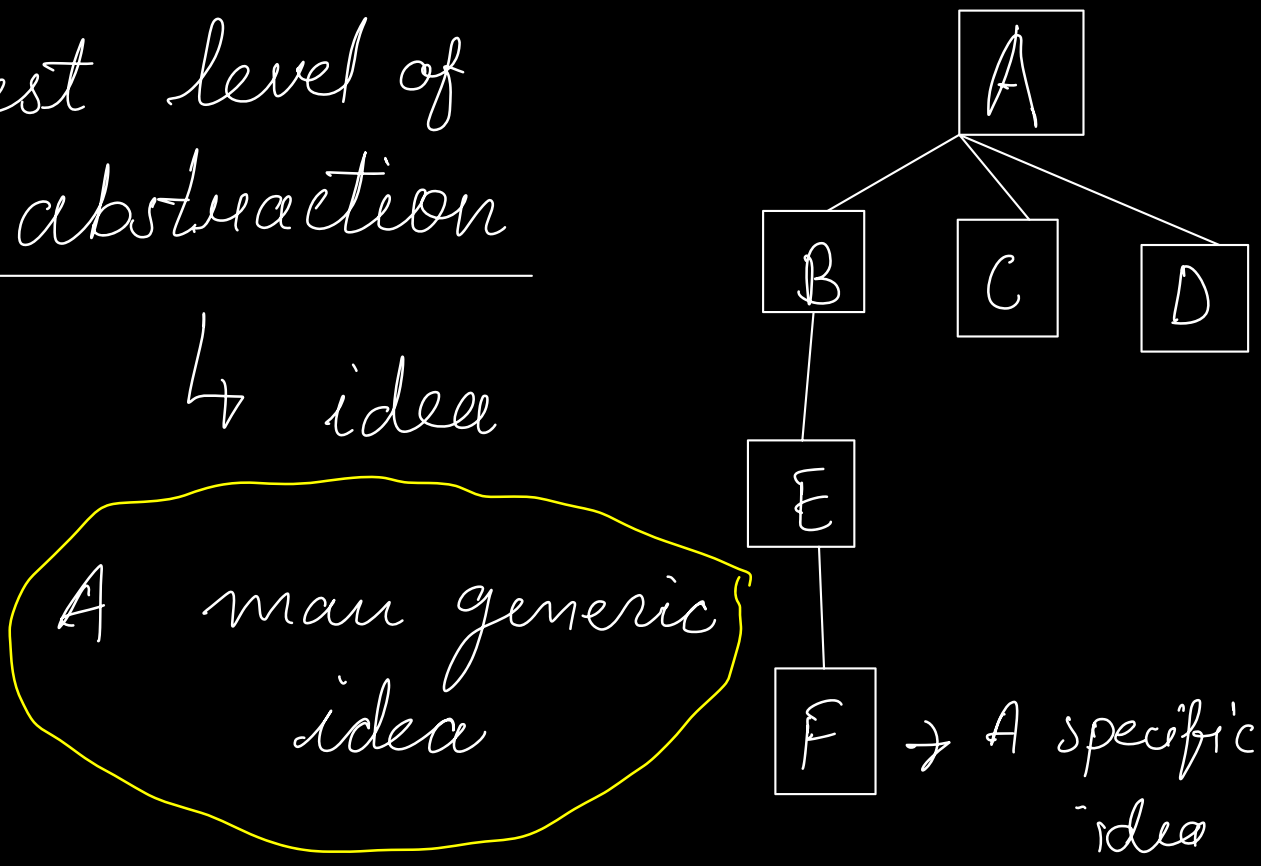


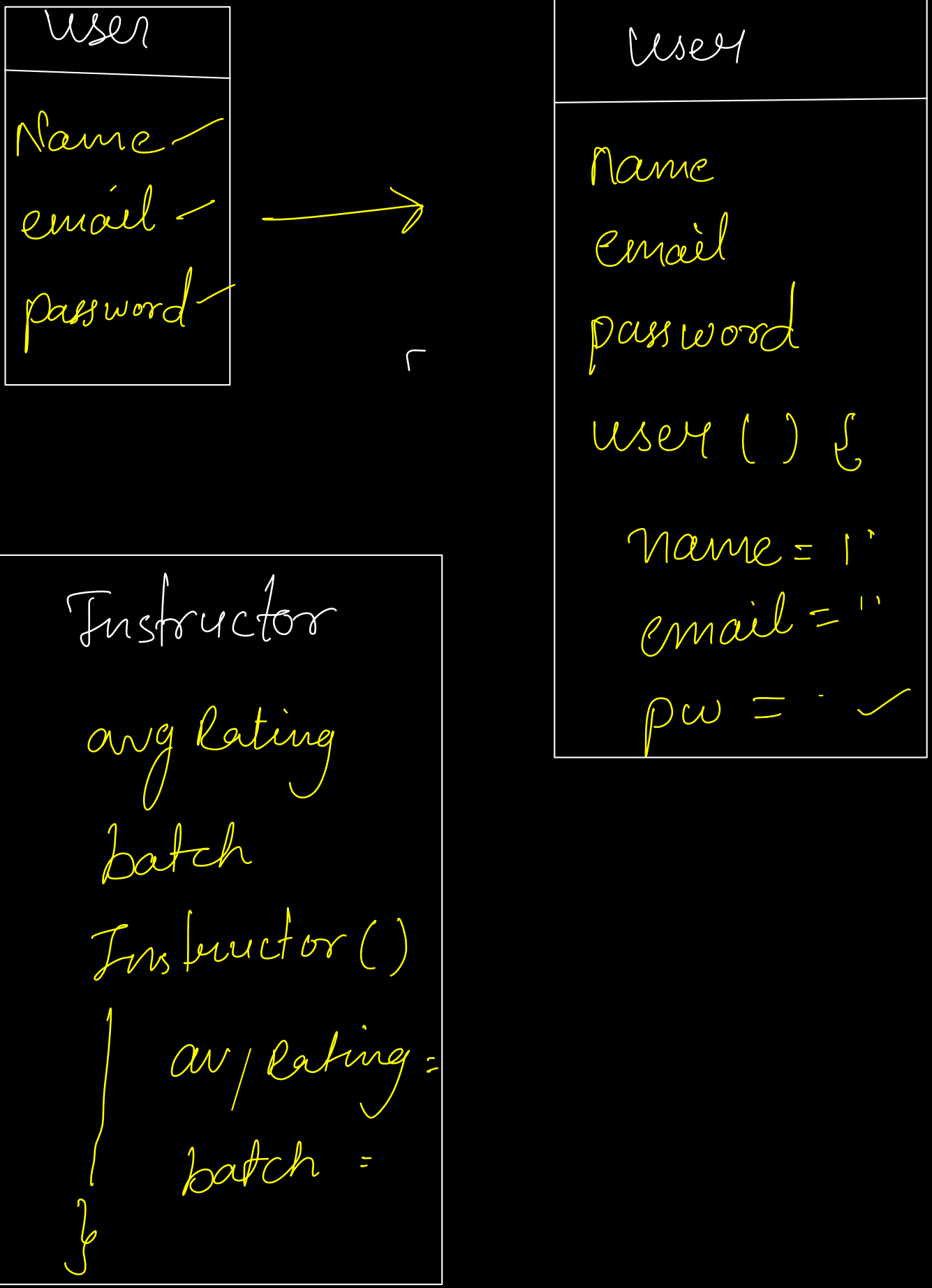
beteh specific type of users

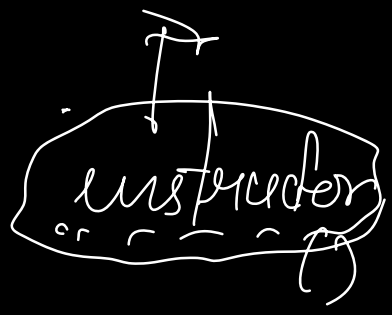
A child class inherits all the members (attributes + behaviours) of the parent class & may add their own members as well.

Highest level of abstraction

↳ idea



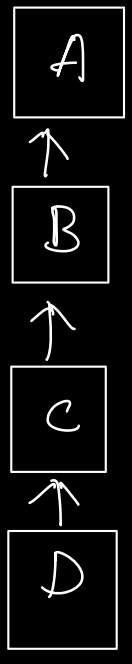

User
Name
email
password
User
Name
email
password
User ( ) {
name = ''
email = ''
pw = ''
Instructor
avg Rating
batch
Instructor ( )
avgRating =
batch =
}

Instructor i = new 

i. userName = ;
i. email = ;
i. pw = ;

Constructor chaining :-

D d = new D( ) ;



1) constructor of D will be called
2) Since D is a child of someone so before execution, it will the constructor of C
3) Similarly, C will call constructor of B
4) B will call constructor A.

A() → B() → D()

Constructor chaining

→ Order of execution of constructors.

Polymorphism

many forms : someone who have multiple forms.

User had multiple forms

Instructor is a User.

Mentor is a User

TA is a user

Student is a user.

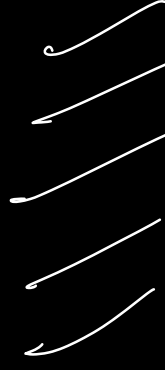
list <user> users =

child

new User();
new Student()
new Instructor()
new Mentor()
new TA()

};

printUserNames(list<user>);



a.user.Name;

Animal a = new Dog()

Data type

Animal ↑ extends Dog

we can put an object of child class in a variable that takes parent type

Dog d = new Animal()

A {
- age
- UserName

}

B extends A
- UnivName

}

C extends B {
- CompName.

}

A a = new C ();

Datatype of a = A

Object gets Created at run time

a.age ✓ a.compName = "ABC" ✗ (throw an error)

a.userName ✓

↓

Compile time error

Compiler allow you to access members of data-type of variable

Types of Polymorphism :-

1) method Overloading :-

Class A {

```
void hello () {
    S.out ("hello world");
}

void hello (String s){
    S.out ("hello "+s);
}
```

hello(): hellow world

hello("vmang"); hellow vmang

Compile time polymorphism

Method overloading

(a) void printHello() { } ✓
void printHello(String s) { }

(b) void printHello(String s) ✓ ph(String)
void printHello(int x) ✓ ph(int) ✗

(c) void printHello(String s)
String printHello(String a)

printHello(String) ↗ printHellow(String)

Not even a valid code

Compile time even not method overloaded

methods are known to be overloaded when you have same method name but different

method signature

↓

Name of method (data types of params)

Same name but different set of params

fn(int, strings) ≠ fn (String, int

2) Method overriding :

```
Class A {
    void dosomething (String a) {
    }
}
```

```
Class B extends A {
    String dosomething (String s) {
    }
}
```

```
Class B extends A {
    void dosomething (String a) {
    }
    String dosomething (String s) {
    }
}
```

Same name

same Signature

✗

Not even a valid code

```
B b = new B()
b.dosomething("Vmang")
```

compile time Error

```
Class A {
    void dosomething (String a) {
    }
}

Class B extends A {
    void dosomething (String c) {
    }
}
```

when parent & child have same method with same name, same signature & same Return type

```
Class A {
```

void dosomething (String a) {
    S.o.p ("Hello")
}

method overriding ??

class B extends A {
    void dosomething (String s) {
        S.o.p ("Bye")
    }
}

A a = new A ()

a.dosomething () // "Hello"

A a = new B () //

a dosomething () // ("Bye")

↳ Runtime Polymorphism