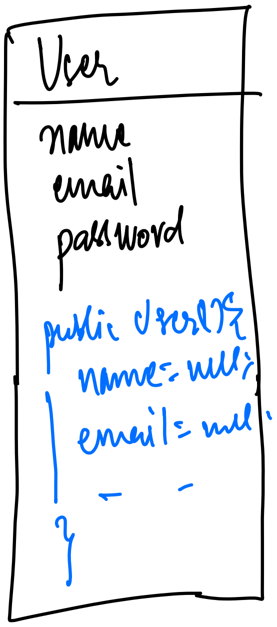
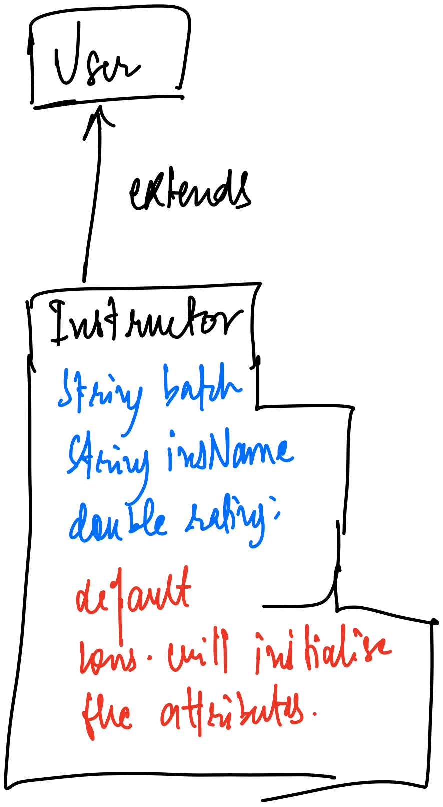
Agenda

1. Inheritance
2. Polymorphism
3. Interfaces.



```
User
name
email
password
public User(){
  name=null;
  email=null;
  - -
}
```

User user = new User();



```
User
  ↑ extends
Instructor
String batch
String insName
double rating;
```

default cons. will initialise the attributes.

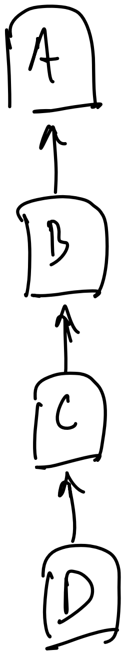
Instructor i = new Instructor();

①
```
class Instructor {
    - -
    -
    -
    public Instructor () {
        name= 'I';
        email= 'J';
        pass = 'K';
    }
}
```

→ this is not ideal

② A constructor of parent class knows better about how to initialise the values of it's attributes.

Steps to Create an object of a child Class.



```
D d = new D();
```

1. Constructor of D will be called.
2. Since D is child of C, it will call the constructor of C even before executing itself.
3. Similarly, C will call the constructor of B.
4. B will call A's constructor.

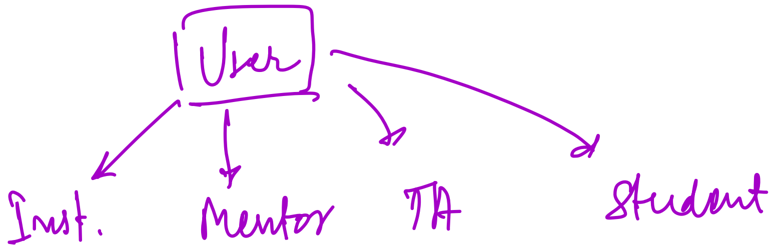
D → C → B → A (calling order)

A → B → C → D (execution order)

Polymorphism

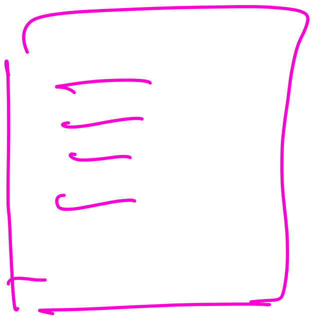
Poly → many

morphism → forms. → something / someone who has many forms.



Client is connected with only the generic properties and not the specific ones.

animal = @703.



① Animal a = new Dog();

Actual object

② Dog d = new Animal(); → ✗

Note:- I can put object of a child class in a variable that takes parent's data type.

A, B and C

```
A {
    int age;
    String name;
}

B extends A {
    String univName;
}

C extends A {
    String companyNam;
}
```

compiler looks at the data type of variable.

A a = new C(); → A a = getObject(); → Compilation Error.

a.companyName = "ABC"; → Failure ✗

a = @700.

→ @700

```
age=0;
name=null;
company=null;
```

* Compiler only allows you to access the attributes of the data type of the variable.

↓

A

changePassword(); →

```
User {
    changePassword(){
    }
}
```

The more generic my code is, the better the reusability.

↳ Use case of Polymorphism.

```
List<Integer> list = new ArrayList<>();
List<Integer> list= new LinkedList<>();
```

Types of Polymorphism

① Compile time

② Runtime

Method Overloading

```
class A {
    void hello () {
        sout ("Hello World");
    }
    void hello (String Name) {
        sout ("Hello" + name);
    }
}
```

This is called method overloading.

```
hello ();
hello (String name);
```

} One method has multiple forms.

Method Overloading is an example of Compile time Polymorphism.

Q1

void printHello(); → printHello( )

void printHello(String s); → printHello (String);

Q2

void printHello (String s); → printHello (String)

void printHello (Integer s); → printHello(Integer)

Q3

void printHello (String s1); → print Hello(String)

void printHello (String s2); → printHello(String) ✗

Method signature :- Name of method (Data type of params)

void printHello (String name, int age)

printHello (String, int);

Rule :- Methods are said to be overloaded, when they have same name and different method signatures.

Method Overriding.

```
class A {
    void doSomething (String a){
        print (Hello);
    }
}
```

```
class B extends A{
    void doSomething (String c){
        print (Bye);
    }
    //Coming from parent class.
    void doSomething (String a){
        -- -
         -
        --
    }
```

If parent and child classes have the same method with same name, same return type and same method signature.
Method is overrided.

This is method overriding.

```
class A {
    void doSomething (String a){
        print (Hello);
    }
}
```

```
class B extends A{
    void doSomething (String c){
        print (Bye);
    }
```

```
Client {
      psvm() {
            A  a = new A();
               a.doSomething();      //Hello

            a = new B();
            a.doSomething();      // Bye
      }
}
```

Method Overriding is run time Polymorphism

```
A {
    dosomething (){
        hello;
    }
}
```

```
B extends A {
    dosomething(){
        Bye;
    }
}
```

```
C extends A {
    dosomething (){
        hi
    }
}
```

```
list <A> {A(), B(), C(), A()}

for obj in A;
    obj.dosomething ();
```

hello, bye, hi, hello

Compiler relies on the data type of the variable, runtime relies on the actual object created