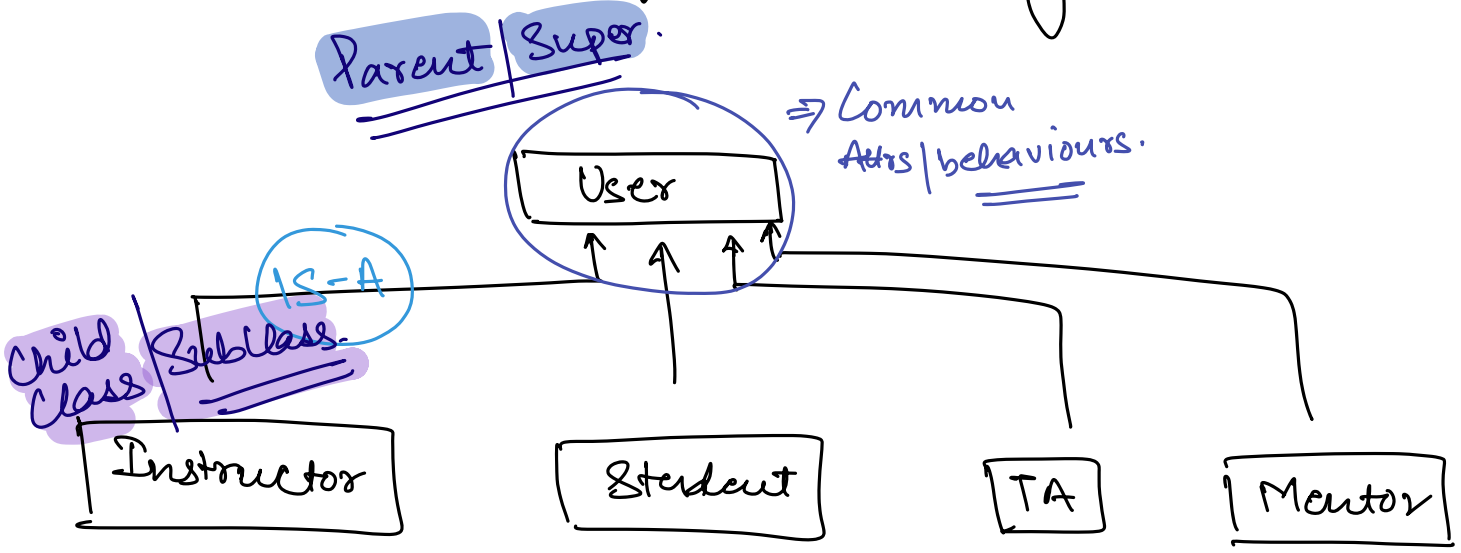
Agenda.

→ Inheritance

→ Polymorphism.

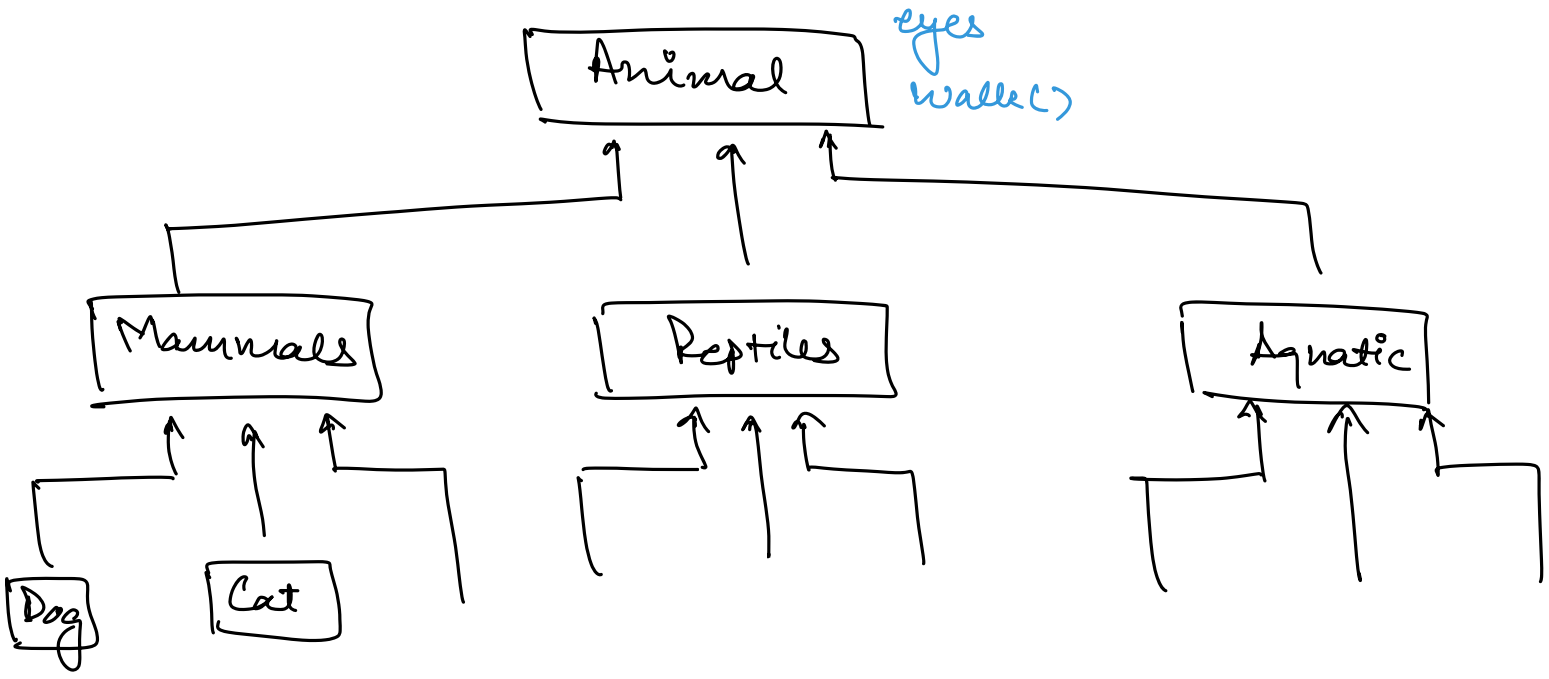
INHERITANCE.

⇒ In real life, we form hierarchy b/w the entities.



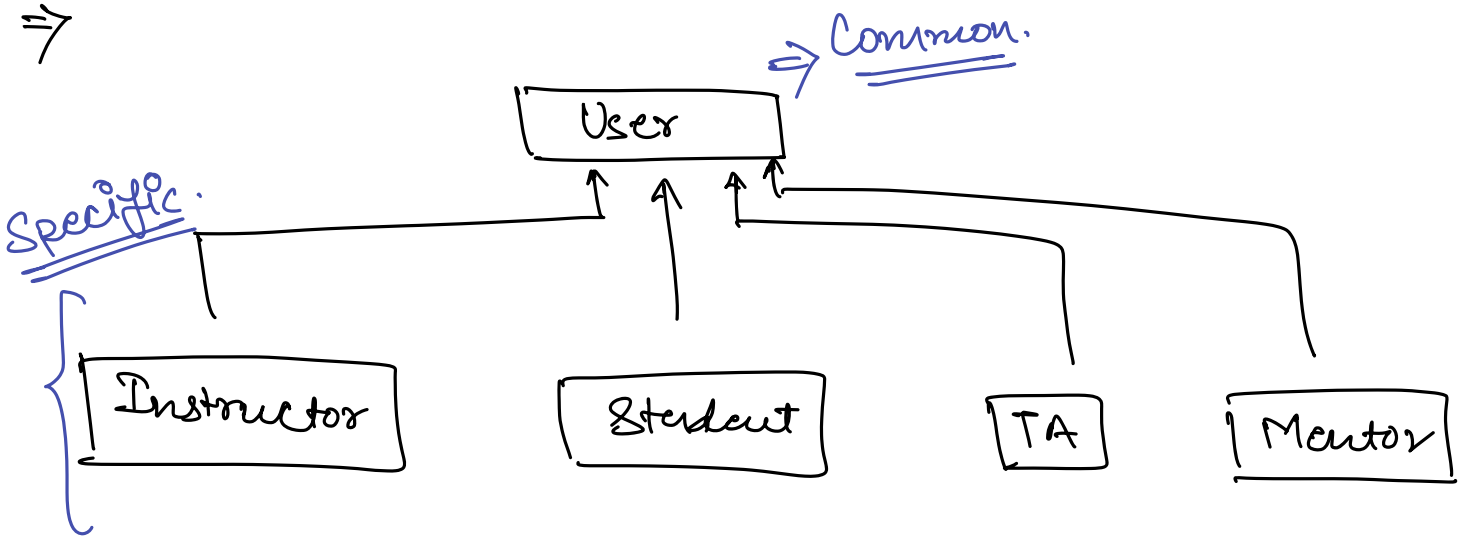
⇒ OOPS allows us to form hierarchy b/w the objects.

⇒ Inheritance.

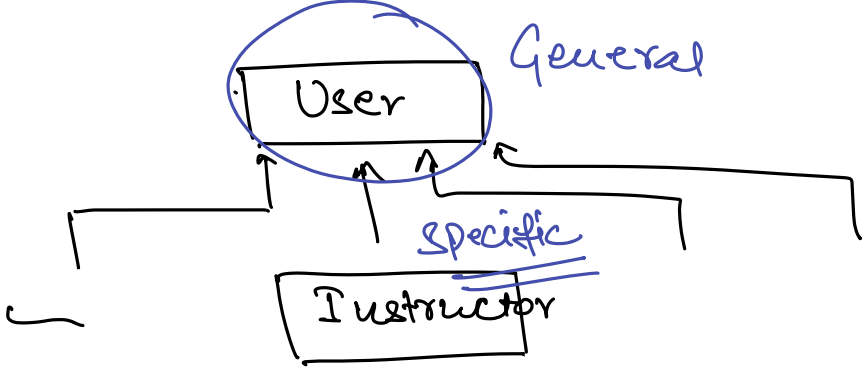


⇒ Hierarchy b/w the entities allows us to share attributes | behaviours.

⇒



⇒ Child inherits all the members from their parent, they may or may not have their own attributes.




```
Class User {
    email
    Password
}
```

```
Class Instructor extends User {
    String batchName
    double rating;
}
```

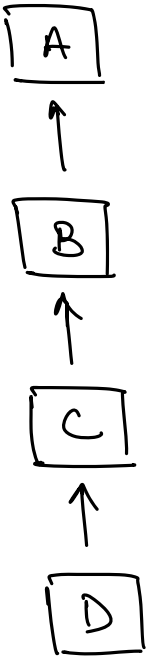
```
Instructor i = new Instructor();
i.email = ____
i.password = ____
```

⇒ How the Object of Child class is created?

When we create an Object of Instructor, somebody should initialize the parent Class attrs. as well.

⇒ Parent Class Constructor.

⇒ Assume: Only default constructors.

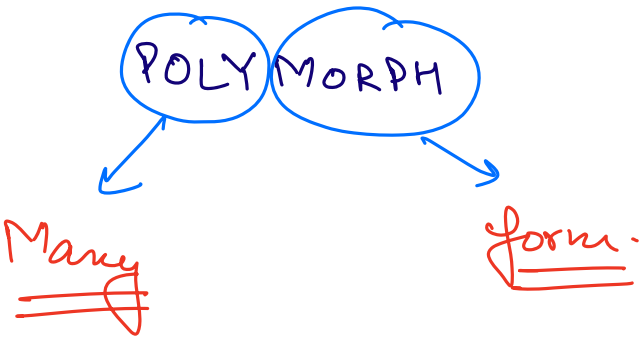


⇒ D d = new D()

1) Calling Constructor of D
2) D will call Constructor of C
3) C will call Constructor of B
4) B will call Constructor of A.

⇒ Whenever we create an object of class, before initialisizing its attrs it calls the default constructor of its parent.

⇒ Polymorphism.



⇒ Something that has many forms.

⇒ User has multiple forms.

→ Instructor is.a User

→ TA _____

→ Mentor _____

X

```
changePassword(Instructor i, String newPass) {

}

changePassword(Mentor m, String newPass) {


}

changePassword(Student s, String newPass) {


}
```

⇒ Code Duplicacy

Inst Mentor

changePassword (User user, String password){

}

⇒ List<TA> —

List<Mentor> —

List<Student> —

List<Instructor> —

→ List<User> users

⇒ User user = new Instructor()

Indians are allowed

Deepak are allowed

Instructo i = new User()

⇒ Animal a = new Dog() ✓

Dog d = new Animal() ✗

⇒ We can put the object of any child class in the parent class reference.

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

```
Class B extends A {
    String univ;
}
```

```
Class C extend A{
    String company;
}
```

A a = new C(); ✓

run time

Compile
↓
Runtime

a.name ✓

a.age ✓

a.company

↳ Compile Time.

Polymorphism

- Compile time ≡ Method Overloading
- Run time

```
Class A {

    fun() {
        Sout("Hello");
    }

    fun(String name) {
        Sout("Hello" + name);
    }

}
```

```
A a = new A()
a.fun() ⇒ Hello
a.fun("Scaler") ⇒ Hello Scaler.
```

Class A {

```
void fun() {
    Sout("Hello");
}

void fun(String name) {
    Sout("Hello" + name);
}

String fun() {
}
```

⇒

Q.1

```
void fun() {—3
void fun(int) {— 3
```

fun()

fun(int)

Q.2

```
void fun(int x) {—— 3
void fun(String s) {—— 3
```

Q.3

```
void fun(String, int)
void fun(int, String)
```

fun(String)

Q.4 void fun(String) { —— }

int fun(String)

fun(String) ✗

⇒ Method Signature.

Name of the method + Datatype of params.

Q.5 void fun(int x) ⇒ fun(int)

void fun(int y) ⇒ fun(int)

Method Overloading ⇒ Same Name, params should be different

⇒ void fun(String) ✓

int fun(int)

⇒ METHOD OVERRIDING. (Runtime)

```
Class A {

    void dosomething(String s){
    ===
    }
}

Class B extends A {

    String dosomething(String s){
    ===
    }
}
```

⇒

```
Class B extends A {

    void dosomething(String s){
    ===
    }

    String dosomething(String s){
    ===
    }
}
```

X

⇒ OVERRIDING.

```
Class A {

    void doSomething(String s){
        Sout("Hello");
    }
}

Class B extends A {

    void doSomething(String s){
        Sout("Hey");
    }
}
```

⇒ If a child & parent are having a method with same signature & same return type

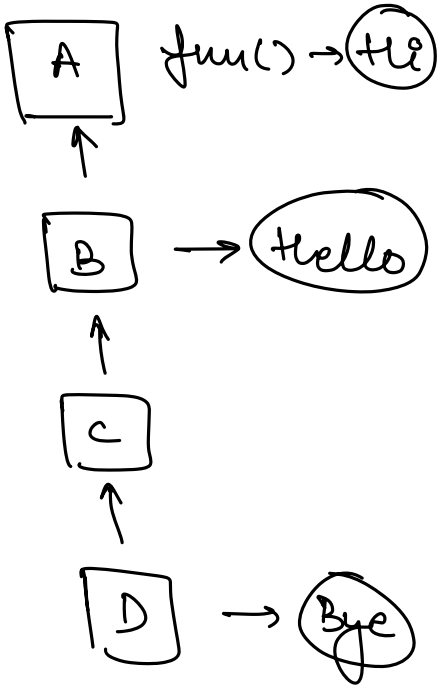
⇒ Method Overriding

```
Class A {
    void fun(String &) {
        sout("Hello");
    }
}
```

```
Class B extends A {
    void fun(String &) {
        sout("Hey");
    }
}
```

⇒

```
main() {
    A a = new A()
    a.fun(); → Hello
    a = new B()
    a.fun(); → Hey
```




```
D d = new D()
d.fun() ⇒ Bye

B b = new D()
b.fun() ⇒ Bye

A a = new C()
a.fun() ⇒ Hello
```

⇒ The method which gets executed will depend on the actual object stored in the reference.

⇒ RUNTIME.