Day 3: Inheritance and Polymorphism

① Inheritance
- what?
- why?
- Types.

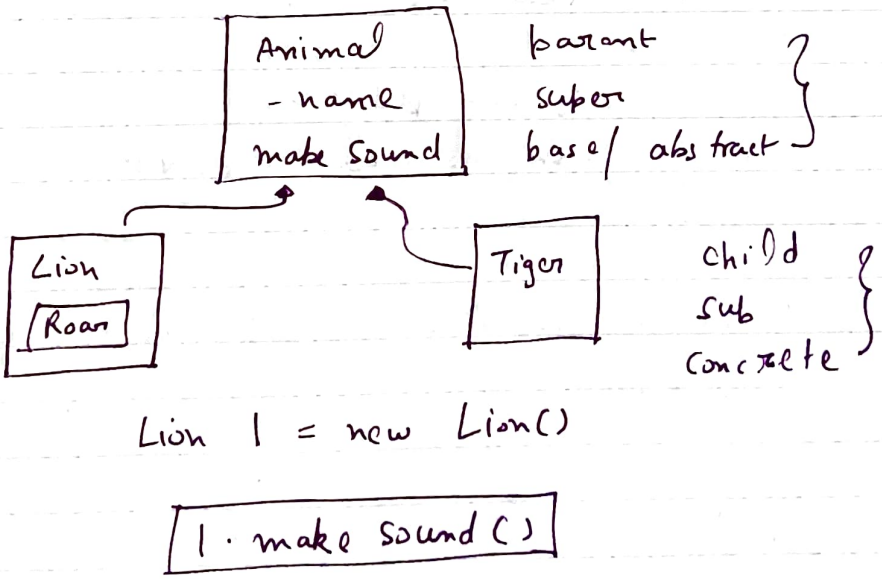
② Polymorphism
- Types.

Inheritance

Student {
id
name
email
PSP
batch
}

Mentor {
id
name
email
experience
}

Field duplication.




```
Lion l = new Lion()
```

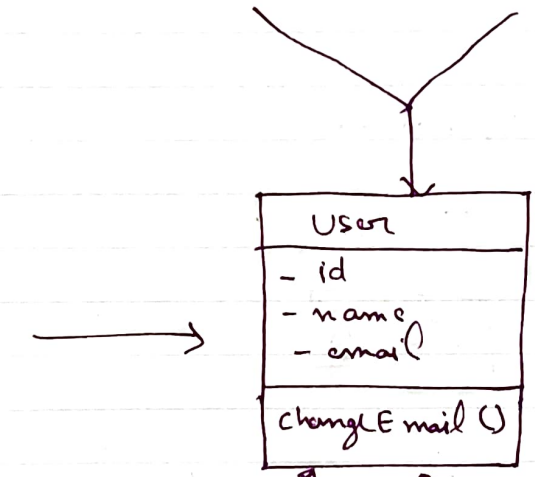
```
l.make Sound()
```

class Student

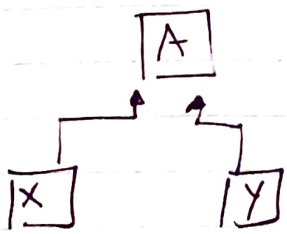
- batch ✗
- attendSession(); ✗
- changeEmail(); ✓

class Mentor

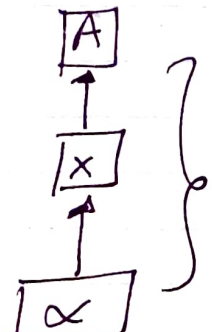
- id
- name
- email
- experience ✗
- guideMentee() ✗
- takeSession() ✗
- changeEmail() ✓



inheritance tree.

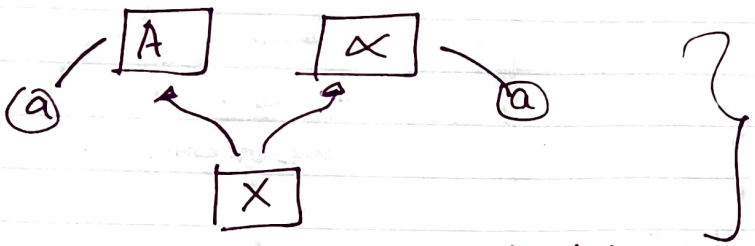


- single level

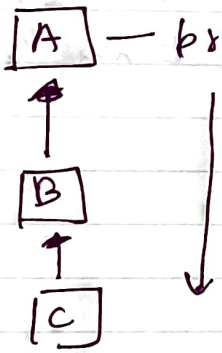


multilevel

Multiple Inheritance → Not in Java
→ Python.



X.a() → Ambiguity.



C.print()

Inheritance — Is-A

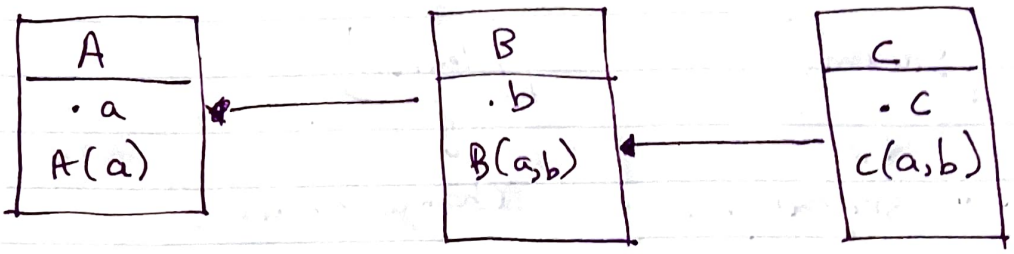
Extends

```
class student extends User {

}

class User extends Animal {

}.
```




```
B(a,b) {
    super(a)
    this.b = b;
}
```

```
C(a,b,c) {
    super(a,b);
    this.c = c;
}
```

Poly morph ism
↳ shape
↳ Many

Many shaped
Animal
↑ ↑
Lion Tiger.

Lion L = new Lion();
↓
Animal

1) sub typing
2) comple time → Method overloading
3) Run time → Method overriding.

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

Map<K,V> map = new HashMap<>();

→ Always write / code to interface → parent class.

Student

new Student().

Mentor

Change email.

② Method overloading

User {

type → student // Mentor

email

psp

batch

experience

User (email, type, psp, batch, exp) {

}.

→ new User (sherlock, student, 1000, 1, null)

→ new User (john, M, null, null, 1)

→ User (email, exp) {

this.email = email;

this.exp = exp;

this.type = mentor;

}

→ User (email, batchName, psp) {

}.

→ Method **Overloading**

```
changeEmail() {
    this.email = null;
}
```

```
changeEmail (email) {
    this.email = email;
}
```

```
a(int a); }
a(int a); }   ×
```

different signatures

① Number of parameters.

```
a(int a)
a(int a, int a)
```

② Datatypes.

```
a(int a);
a(string a);
```

```
Void a(int a)
String a(int b)   ×
```

→ Method **Overriding**

Animal

↑

makeSound → "meow"

↑

Dog

```
Dog d = new Dog()
d.makeSound();
```

diff impl for diff classes

```
class A {

    print () { subject ("A");
}
```

```
class B {
    @ Override
    Print ()  { sysout ("B")
}.
```