OOP - III

OOP
SOLID
Design Patterns

- Constructor chaining
  - User → Student
  - default → implicit
  - para → explicit

* OOP → Python
    - I → Multiple Inhenitance
        → __init__
    - E → ✗
        → _private
        → __scrambled

Polymorphism
→ Subtyping
→ compile
→ run time

→ overriding

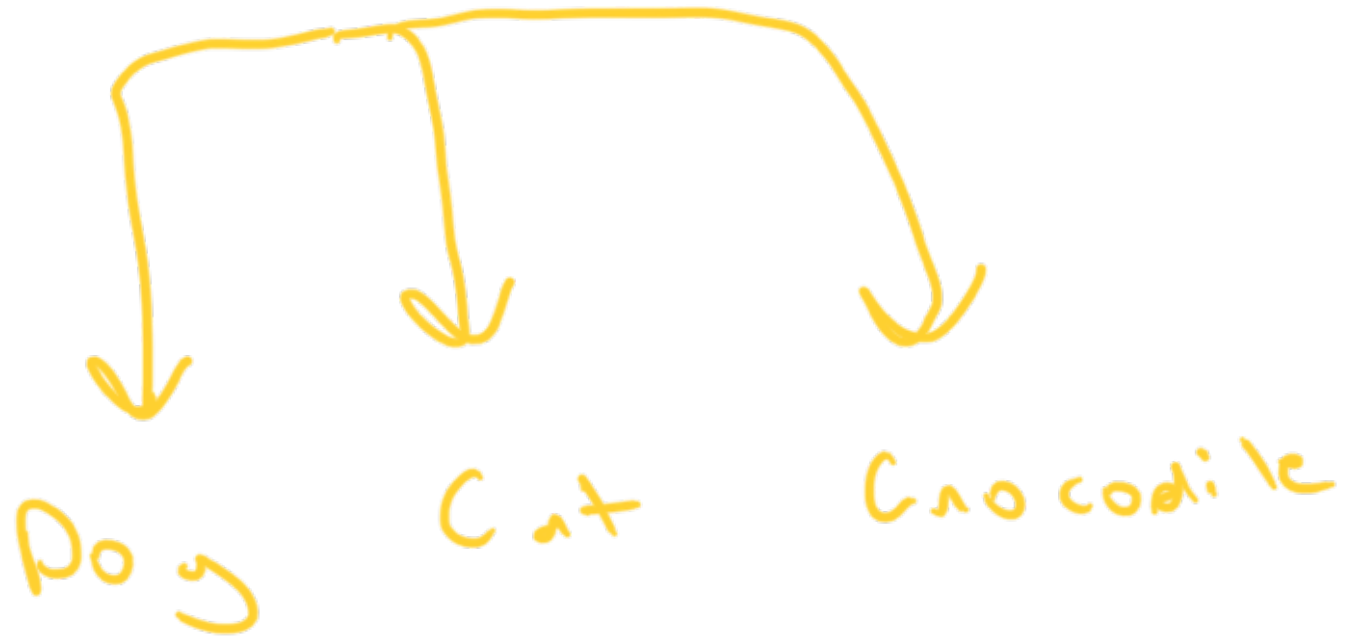
→ overloading

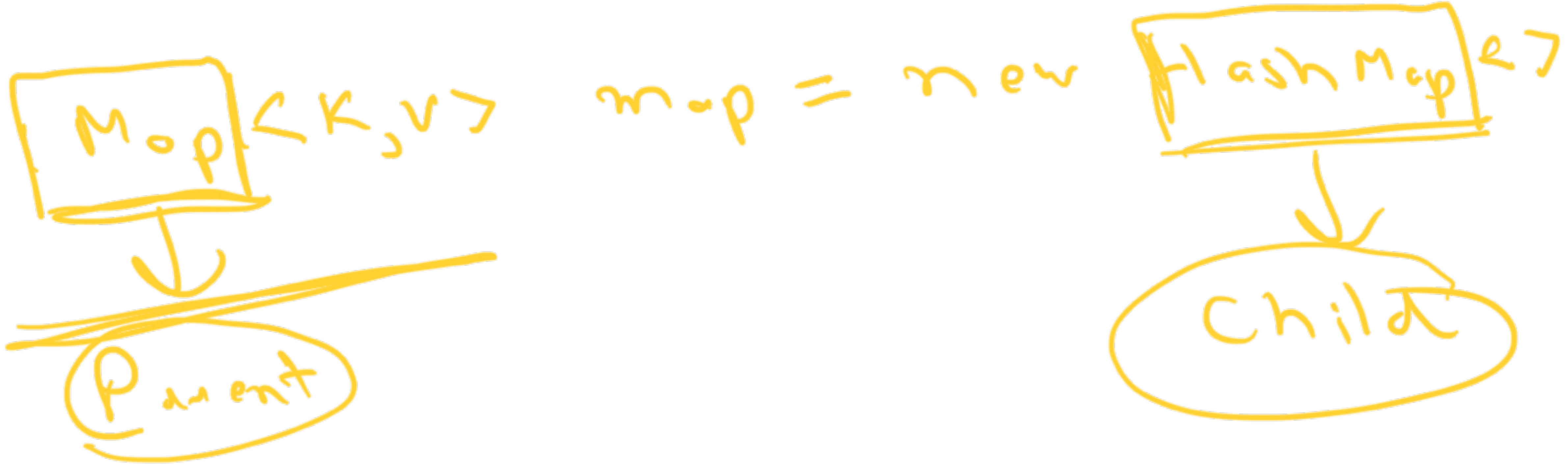
PM

$$\begin{tikzcd}
\text{poly} \arrow[d] & \text{morphism} \arrow[d] \\
\text{many} & \text{form / shape}
\end{tikzcd}$$

$\boxed{\text{Many Shaped}}$

Pet





Child → Parent

subtyping

$$\begin{tikzcd}
 & \text{User} \arrow[r] \arrow[dl] \arrow[d] \arrow[dr] & \text{changeEmail} \\
\text{Student} & \text{Mentors} & \text{TAs}
\end{tikzcd}$$

List < Student> → changeEmail (List<S>)

→ [illegible]

< Mentor >

< TA >

changeEmail(List

```
changeEmail(List<S>, List<M>)
{
        for each student
                changeEmail
        for each mentor
```

changEmail

}

Extensibility $\rightarrow$ Instructors

Student $\rightarrow$ USER

Mentor $\rightarrow$ USER

List < USER >

changeEmail (List<USER>){

for each user

(1) changeEmail

(2) user.psp = 0

child

$$\begin{tikzcd}
\text{Cost} \arrow[r] & \text{child} \arrow[r] \arrow[d] & \boxed{\text{USER}} \\
& \text{State ?} &
\end{tikzcd}$$

behaviour

$\boxed{\text{User.pID}}$

```
Map<>      = new HashMap
HashMap<>  =      "
```

```
User  student = new Student()
```

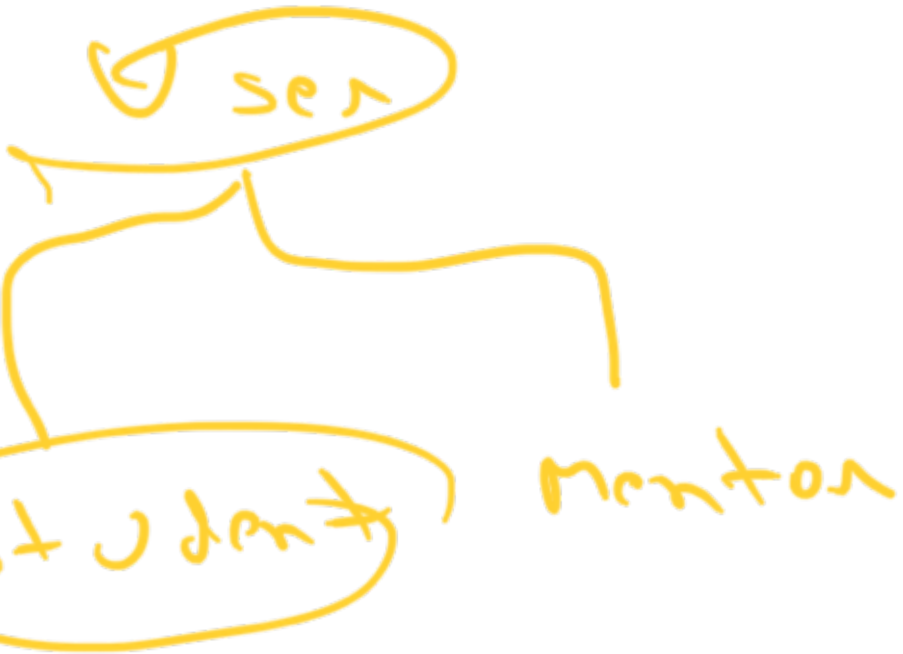
"

Student Student1

student.psp ✗

student2.psp ✓

Many forms



class

Method overloading

ctor

```
class User {
      name;
      email; Optional

  →   User (name, email) {
        { if (email != null)
              { this.email = email.trim(
              }
        }
  →  User(name) {
        this.name = name;
     }
}
```

Method overloading

a(int b)
a(int c)

method signature

→ methodname

→ number of parameters

→ data type of args

a (int a)
a (int a, int b) } ✓

a (int a) ✓
a (String a)
a ( ) ✓

User getUser ($\overset{\text{String}}{\text{none}}$) {
....

```
}

getUser(~~Int~~ id) {

}

~~void~~ getUser (String email) {
        ...
}                                  }

main() {

    getUser("name")               }
```

- getUser("email")

→ return types X

→ arg. names X

→ access modifier

$\underbrace{\text{getUser}}_{}\ (\underbrace{\underbrace{\text{String}}_{}\ \text{name}}_{})$

get~~User~~ (String email)

String

{

("name")

getUser(

getUser(" email")

compile

run time

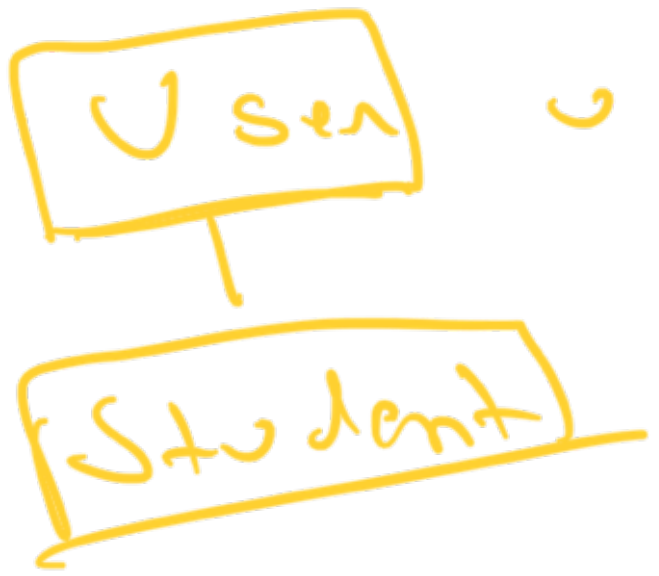
compile time PM

→ overloading

→ ctor

→ methods

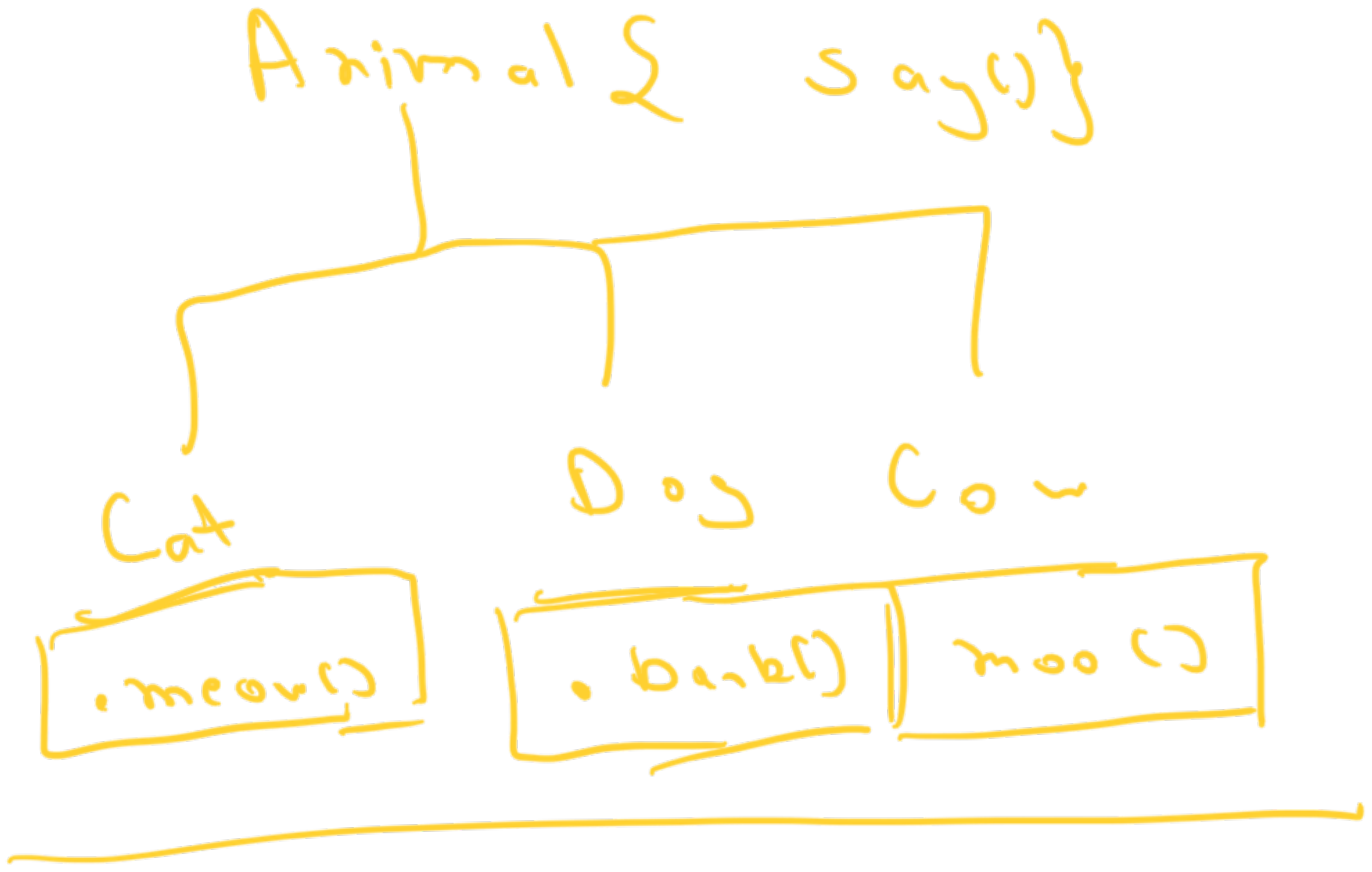
Rutine polymorphism



```
user = new Student();

User {

    printInfo()

}
```

}

student.printInfo()



= new Cow();

```
Animal cow = new

        cow.moo()  X

          Animal
          Say() {
               if ( type == cow)   X   [ OC ]
                         moo
X              elseif( type == dog)
                         bark
        }
```

extensible   Panda

<u>Method overriding</u>

All persons can walk

```
Animal {

    sound() {
        sys.out("No sound")
} }


Dog {
    @Override
    sound() {
        sysout(" Bark")
    }
}
```

User, Student, mentor

Animal, Dog, Cow

Runtime polymorphism

dog. sound()

Runtime → sound()

→ Child → c. sound()

→ Parent → p. sound()

JVM → inheritance chain

1 — 2 — 3 — . . Parent

^ ^ ^ ^

$\boxed{\text{Animal}}$ animal = new Cat()

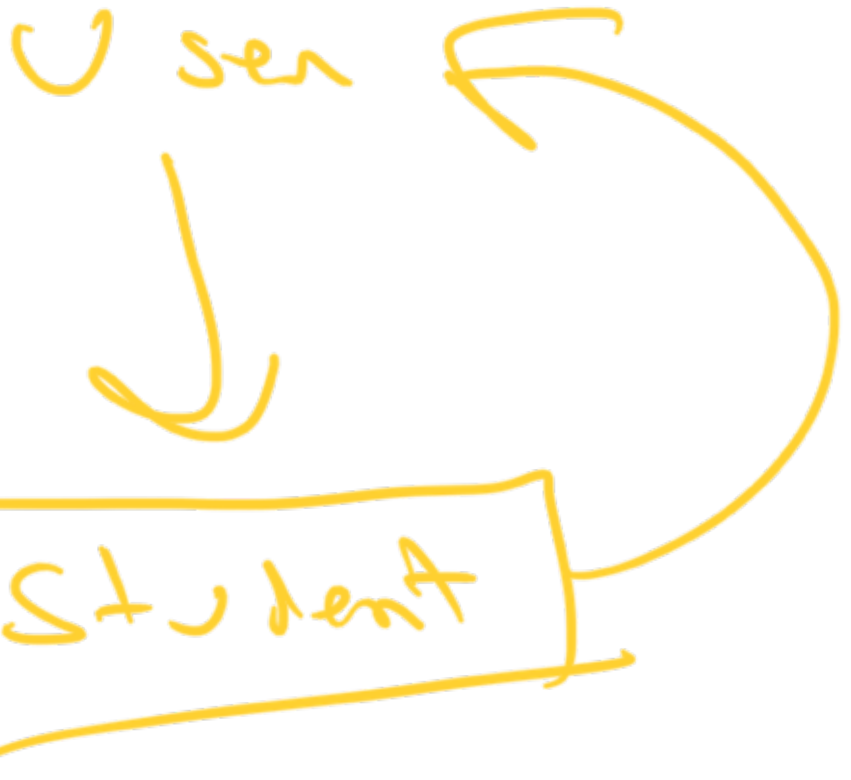
cat.sound()

$\boxed{\text{Cat}}$ $\boxed{\text{cat}}\boxed{\text{.sound()}}$

Ⓐ c = new C()

c.method()

Subtyping



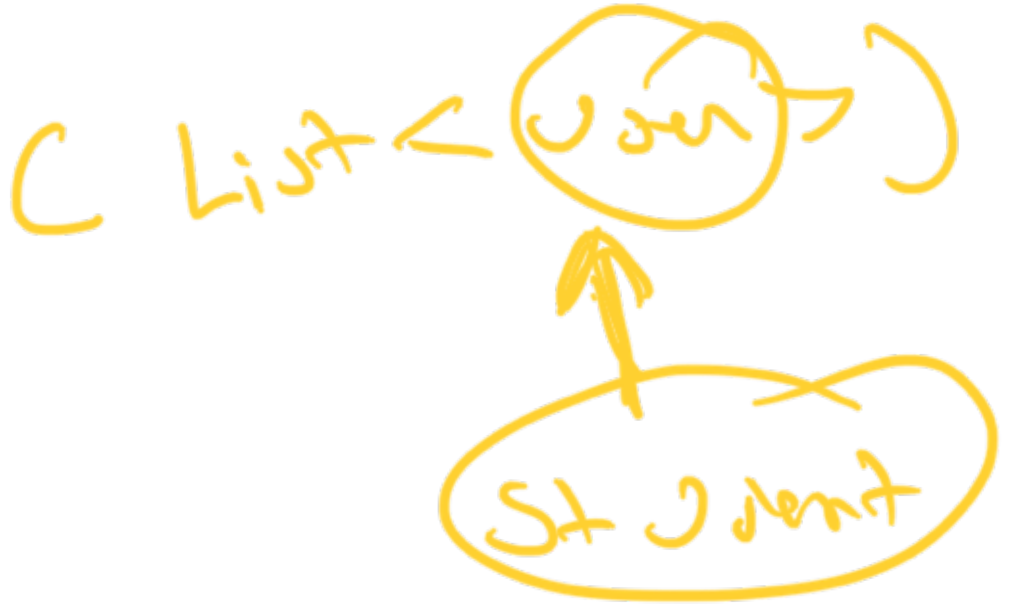
```
Map<K, V> map = new HashMap<>();
```

Map [illegible]

interface
parent

child

→ how do I create a method for all types of Users



subtyping

student.p sp

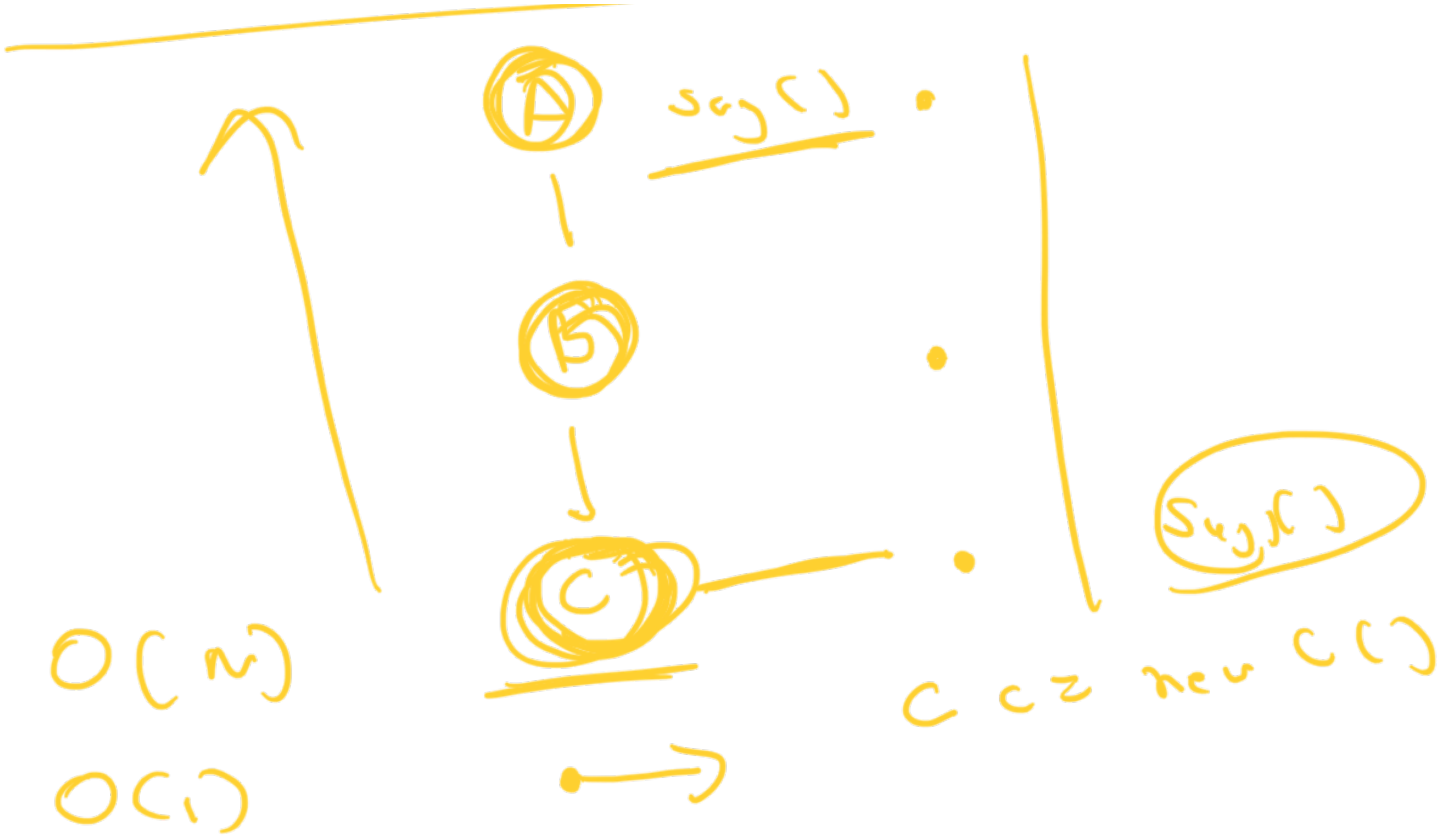
Jun - $\boxed{\{ \quad \}}$

runtime → knows

→ instance of

(Integer) user

→ Class Cast Exception



if child overrides parent

→ only child will be

checked

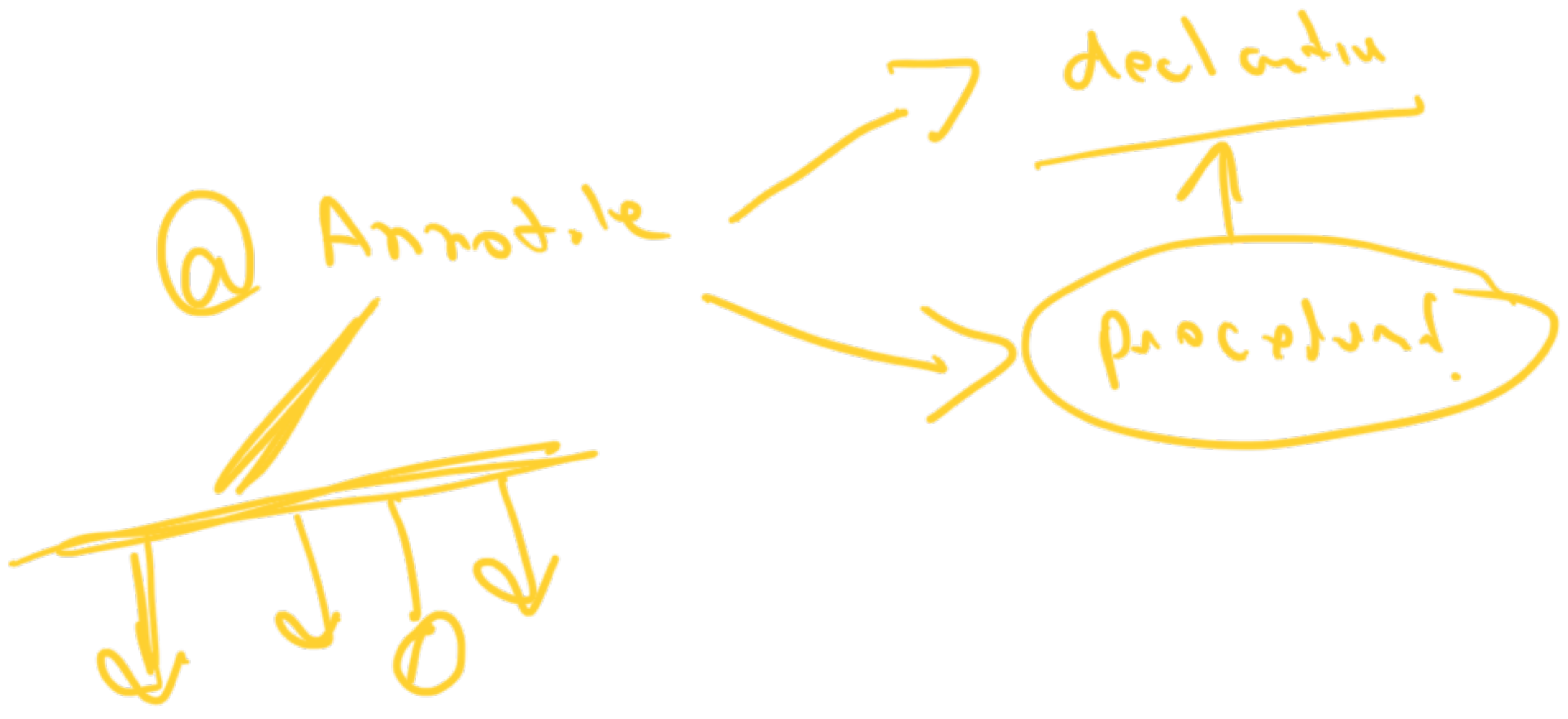
→ use child version

if no class overrides method

→ go through each class

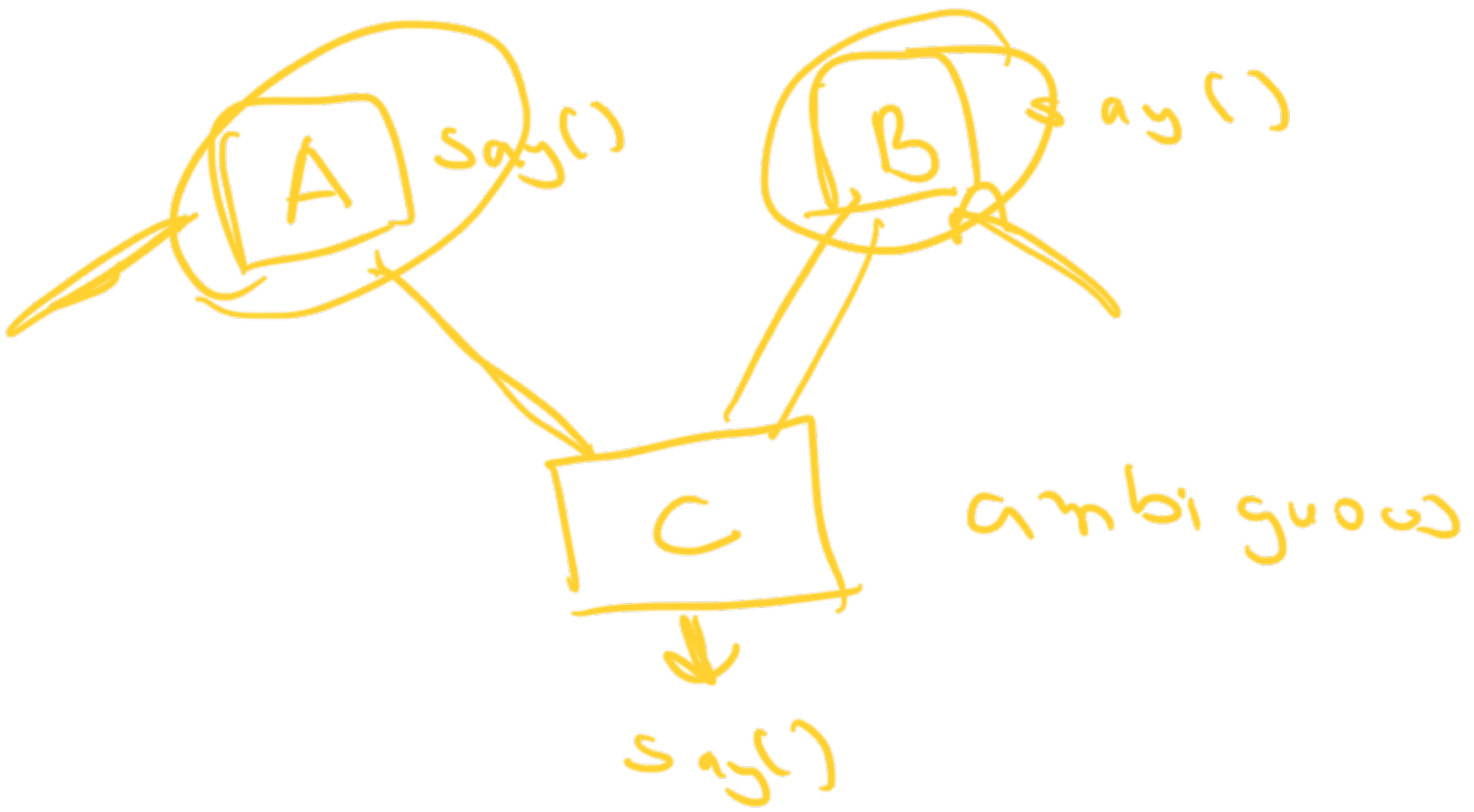
```
A {
    say()
}
```

```
B {
      say() {
          super.say()
      }
}
```



Anno. Process.

Diamond



c. say()

Java → No multiple inheritan..

classes

Python → define

C++ → virtual

Duck typing

Python → lazy

$$\begin{tikzcd}
 & \underline{\text{Animal}} \arrow[dl] \arrow[dr] & \\
\underline{\text{Dog}} & & \underline{\text{Cow}} \\
\text{(Carnivore)} & & \text{(Herbive)}
\end{tikzcd}$$

class Cow extends Animal
Human

class Dog ex A, C

Duck typing → Lazy

Adhoc pm

If it has the method/member
then it is of the same class

```
A. say()

method(A):
    A. say()
```

[illegible]