Remaining OOP

Today

- Polymorphism
- Interfaces
- Abstract Classes
- Static

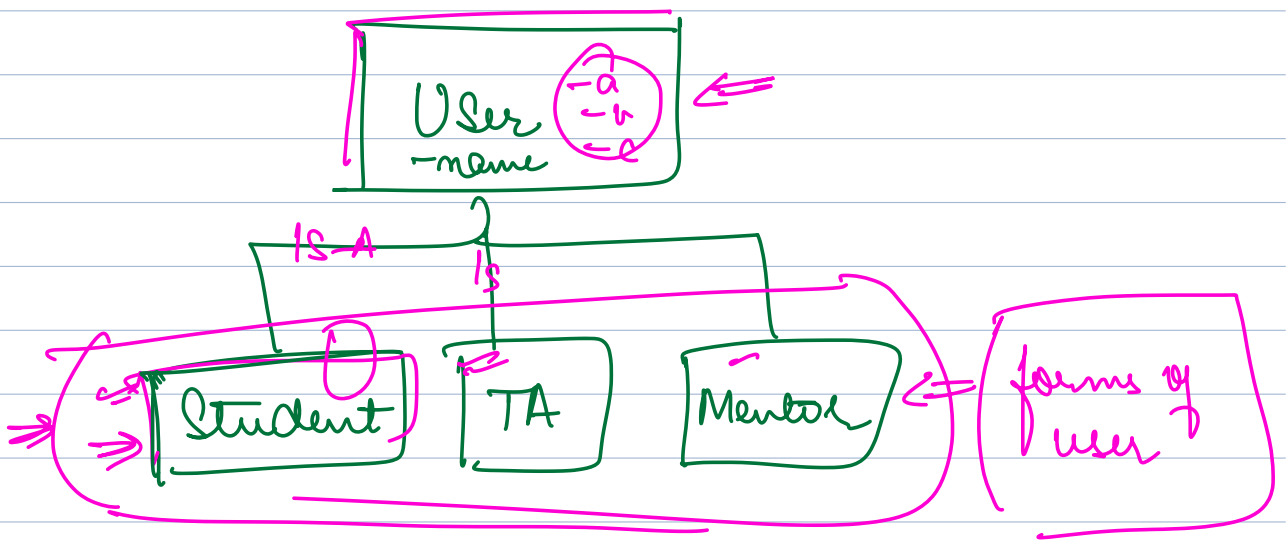
Min

Will start at 9:10PM

POLYMORPHISM

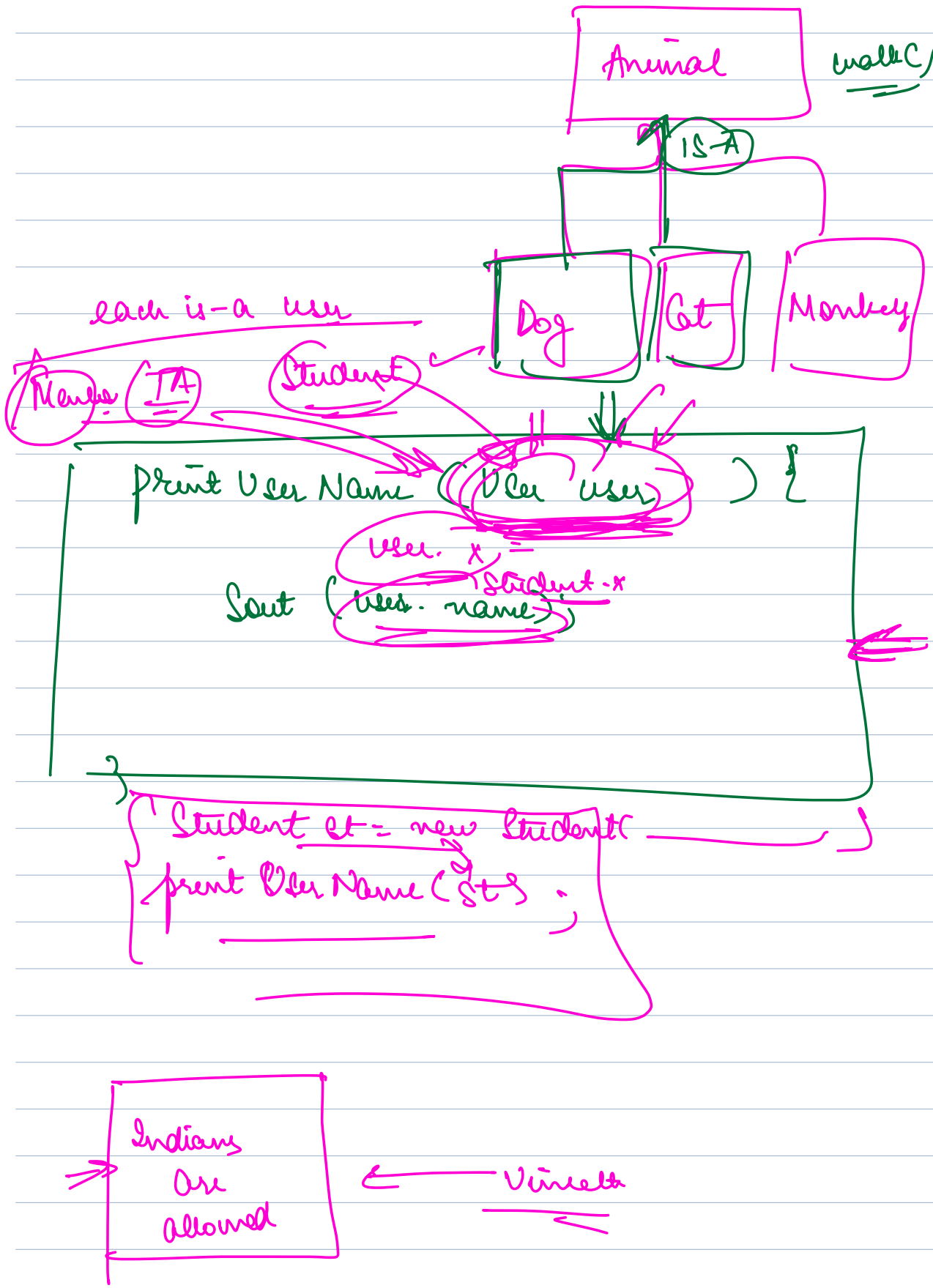
Many Form

↳ used to rep something that has multiple forms.

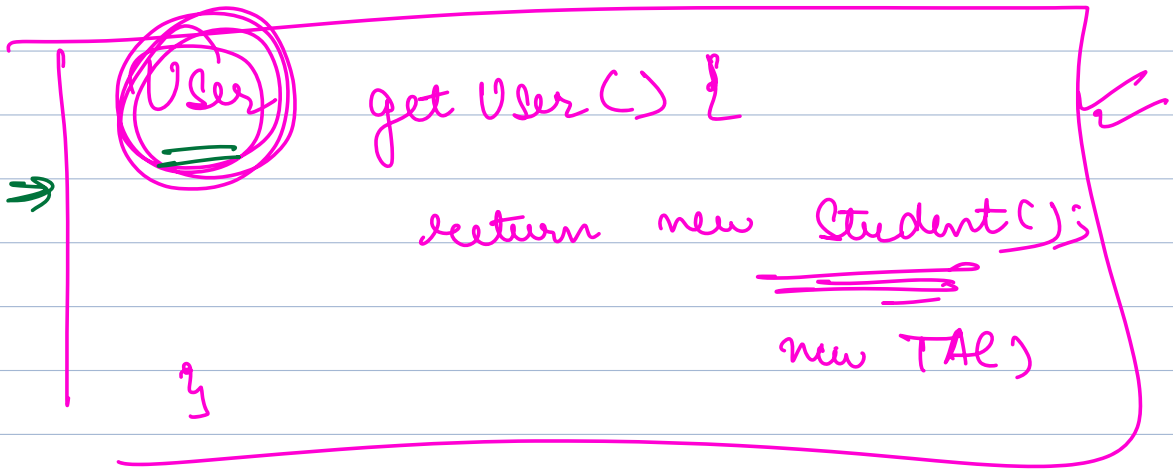


User has multiple forms.

- Student IS-A user
- ta IS-A user
- mentor IS-A user
- instructor IS-A user


Animal
walkC/
IS-A
Dog
Cat
Monkey
each is-a user
Mentor
TA
Student
print User Name (User user) {
user. x =
Student.x
Sout (user. name)
Student st = new Student(
print User Name (st) ;

Indians
Are
allowed
Virat

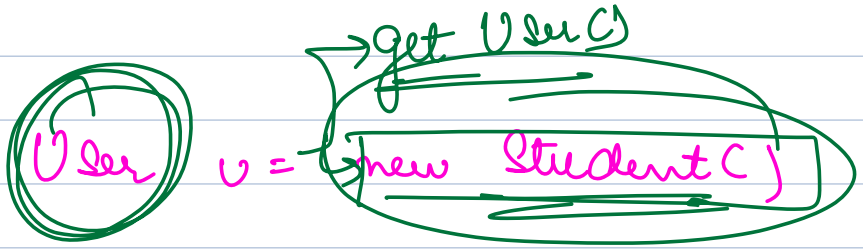


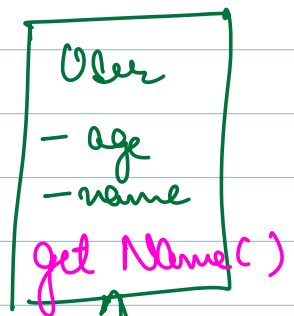
List<User> users {
new User(),
new TA(),
new Mentor(),
new Student(),
}

Restaurant {
List<Indian> indians {
vineeth,
}
}

User u = new Student();

Anywhere where data type is of a parent class (User), you can put objects of any child class of that as well here.





cout (u.name)
cout (u.age)
cout (u.psp)

User, Student, TA, Mentor

User u = getUser();

u.psp;

⇒ Not even compile

Compiler only allows access to properties based on data

type of variable, and not the Real obj.

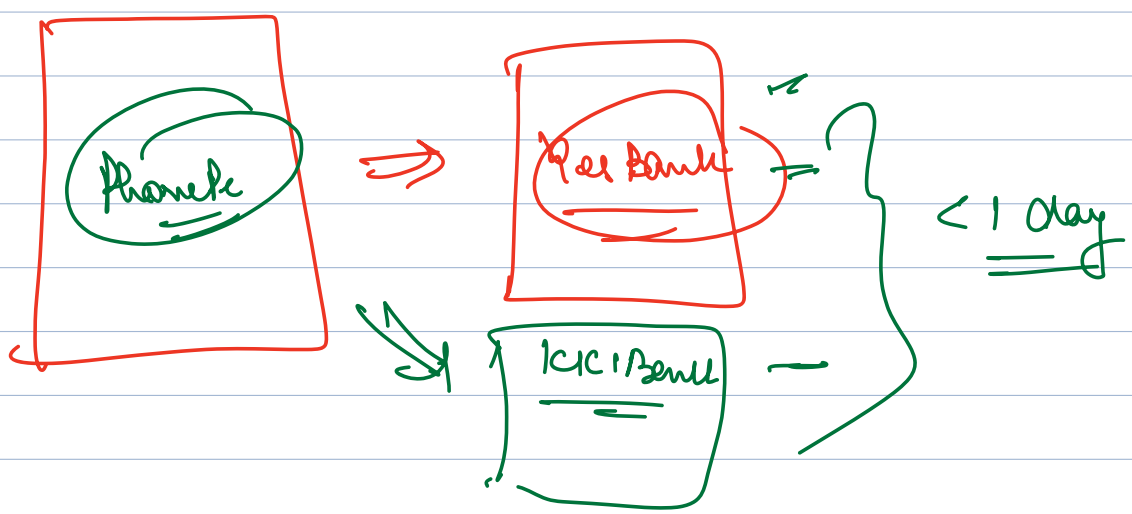
```
getUser() {
    int i = rand() % 2;
    if (i == 1)
        {
    } else if (i == 2)
        {
    }
```

User u = new Student() ✓ first is always a better approach

VS

Student s = new Student()

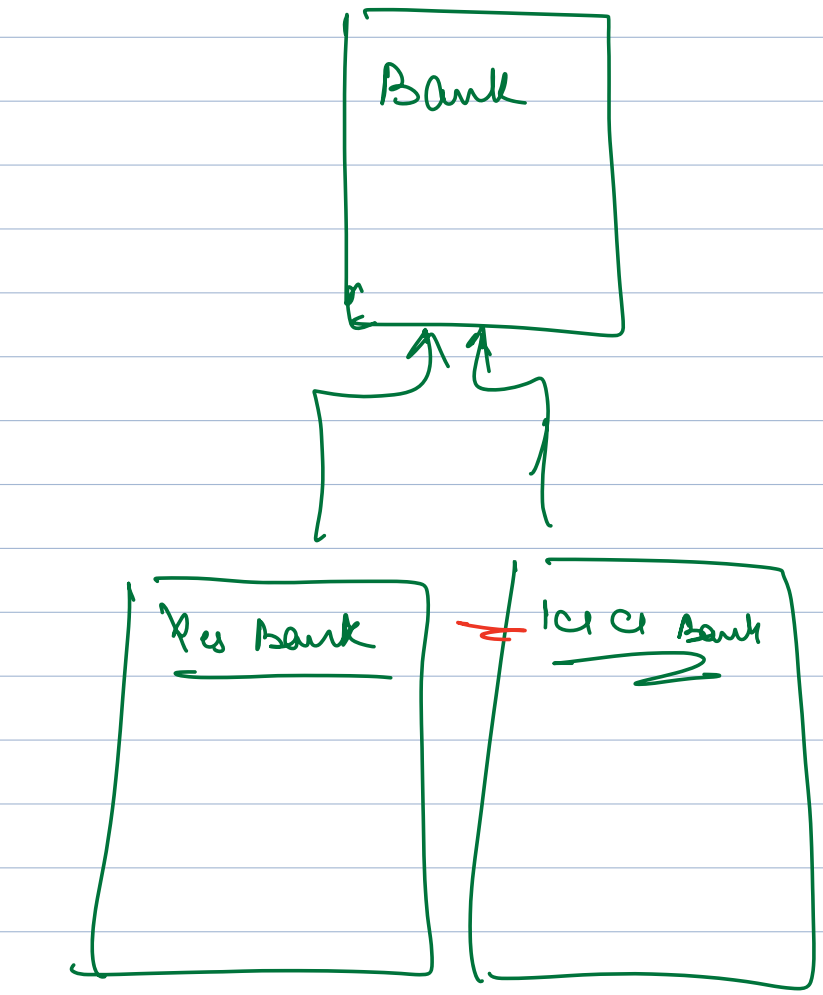
→ The more general your code is, the easier it is to extend.



Example

Yes Bank yb = ____

Testing



Bank b = new YesBank();

new ICICIBank()

changePassword(User u) {

→ u.setPassword(newPassword);

}

changePassword(newStudent)

updateBatch(Student stud) {

}

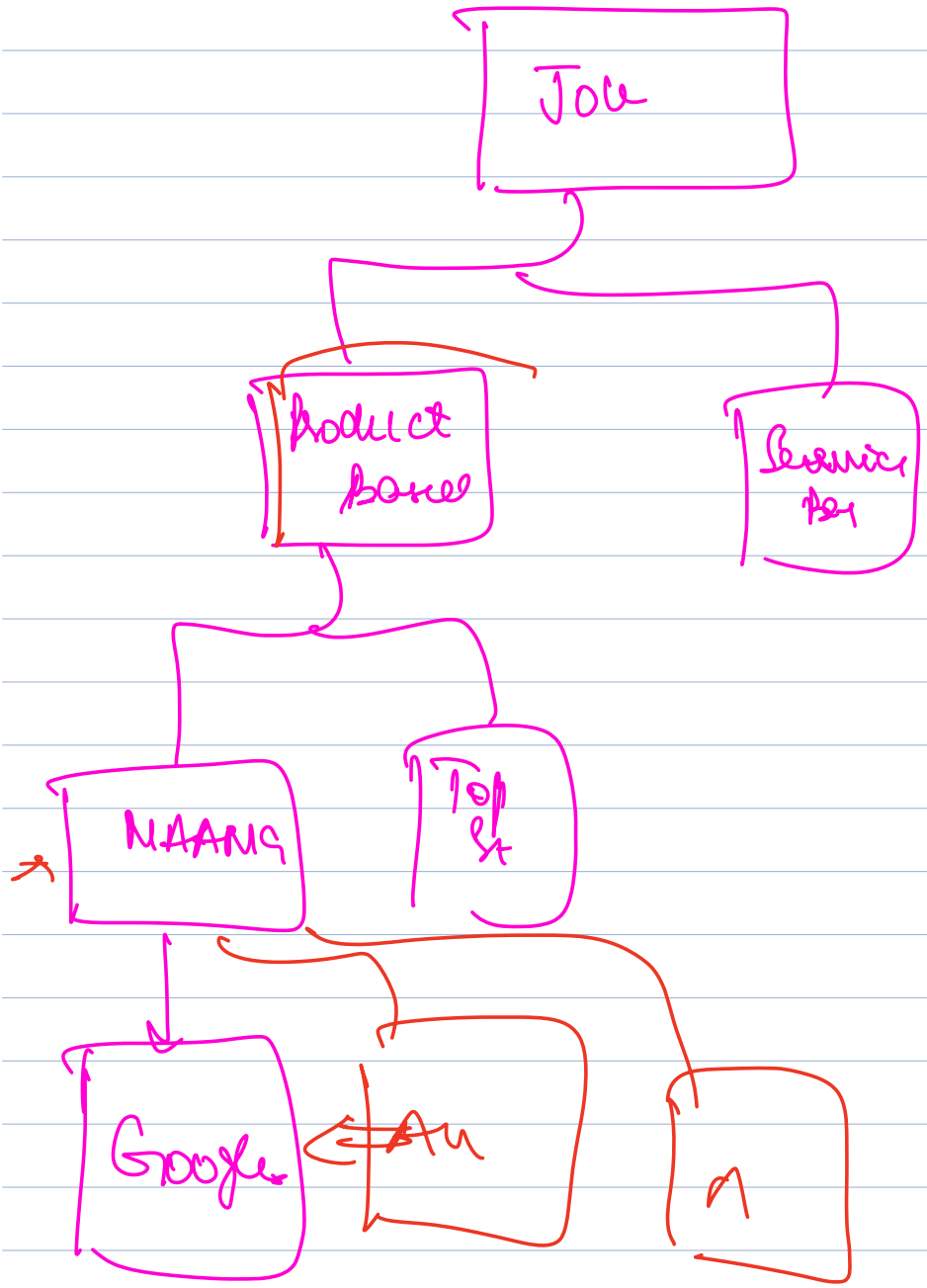
→ It is possible to put obj of child class in variable of parent class.

→ But you will be able to access only attr that are defined in parent

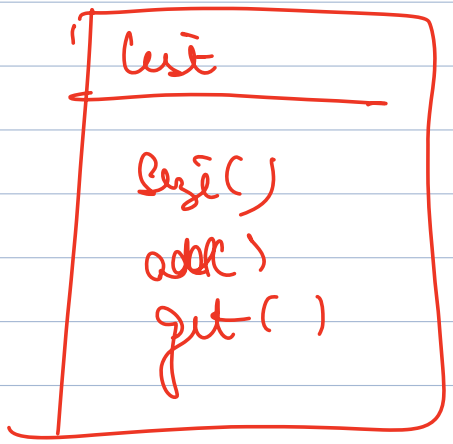
User

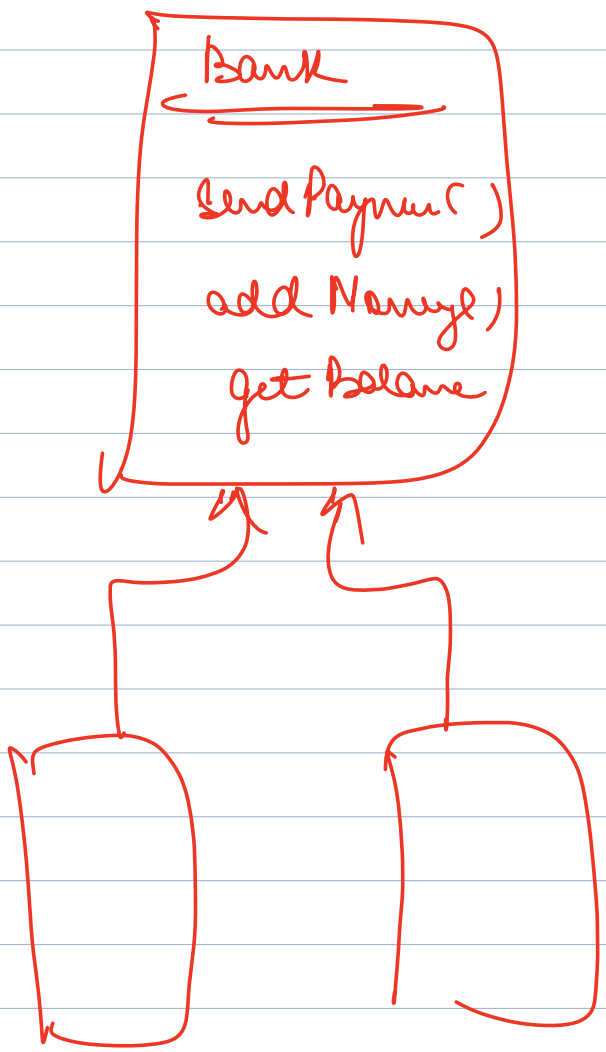
Student.

List<User> = new ArrayList();



Create data type of variable to be as general class as possible.

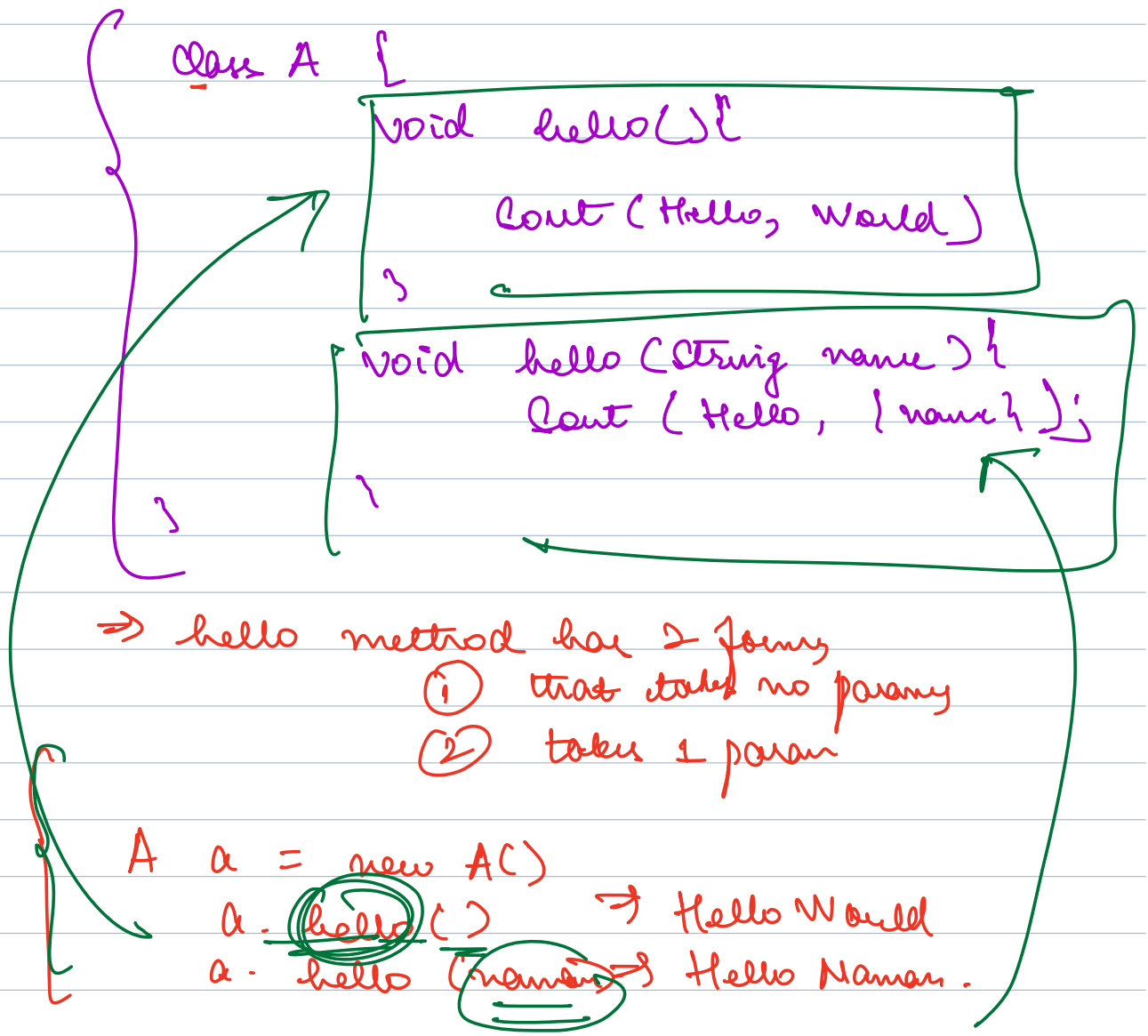

List
Size()
add()
get()


Bank
SendPayment()
addNewAcct()
getBalance

Polymorphism

1. Inheritance
2. Method Overloading
   - ↳ Compile Time Polymorphism

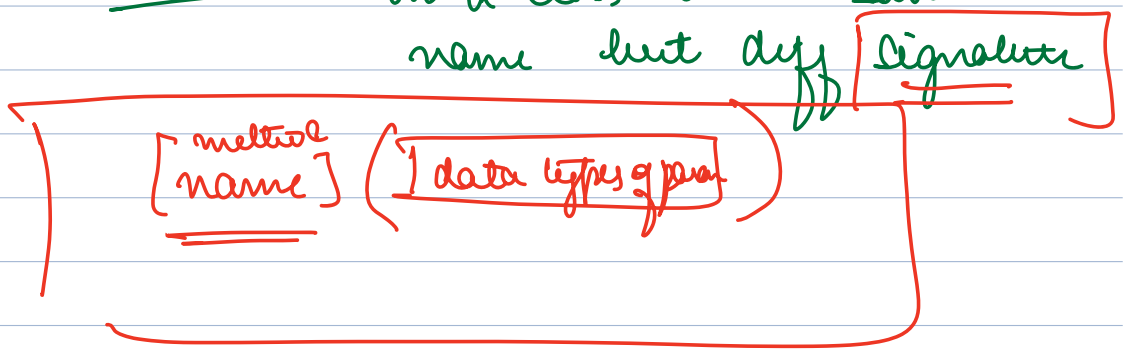
METHOD OVERLOADING

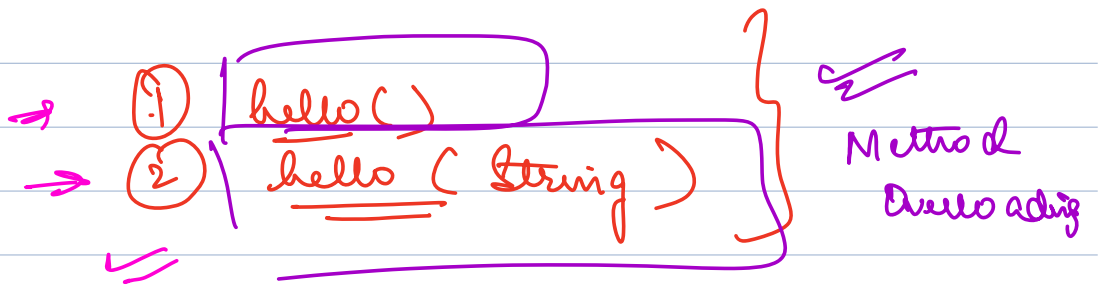


⇒ hello method has 2 forms
1. that takes no params
2. takes 1 param

```
A a = new A()
a.hello()        → Hello World
a.hello(name)    → Hello Naman.
```

Method Overloading : multiple methods in a class with same name but diff Signature



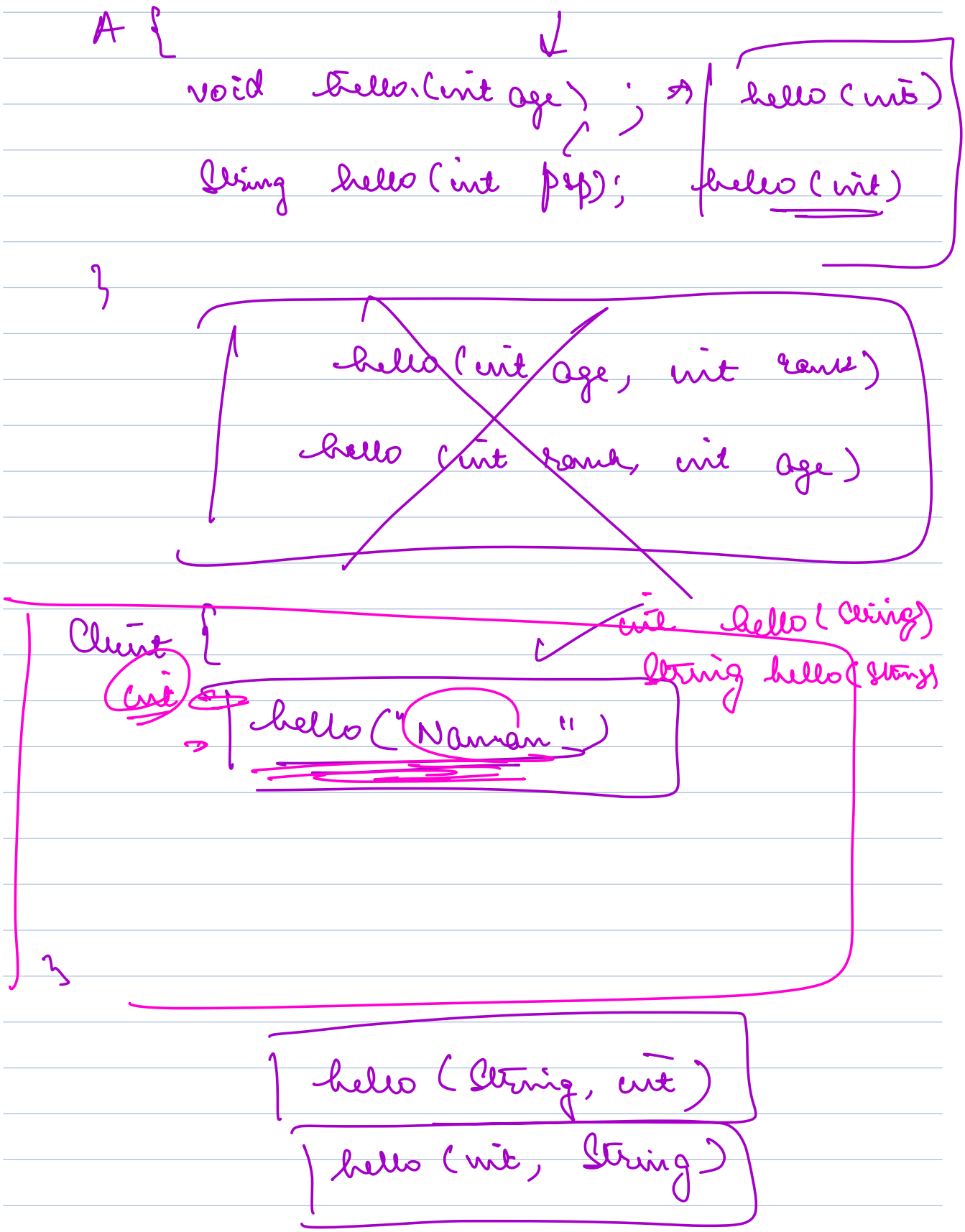


→ 2 methods with same signature can't be there in a class.

A {

void hello (int age) ;	→	hello(int)
String hello (int age) ;	→	hello (int)

}


A {
void hello(int age);
String hello(int pqr);
hello(int)
}
hello(int age, int rank)
hello(int rank, int age)
Client {
int
hello("Naman")
int hello(String)
String hello(String)
}
hello(String, int)
hello(int, String)

hello (User)
↑
hello (Student)

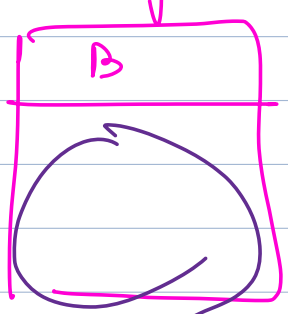
POLYMORPHISM

1. Inheritance
2. Method Overloading
3. Method Overriding

Method Overriding

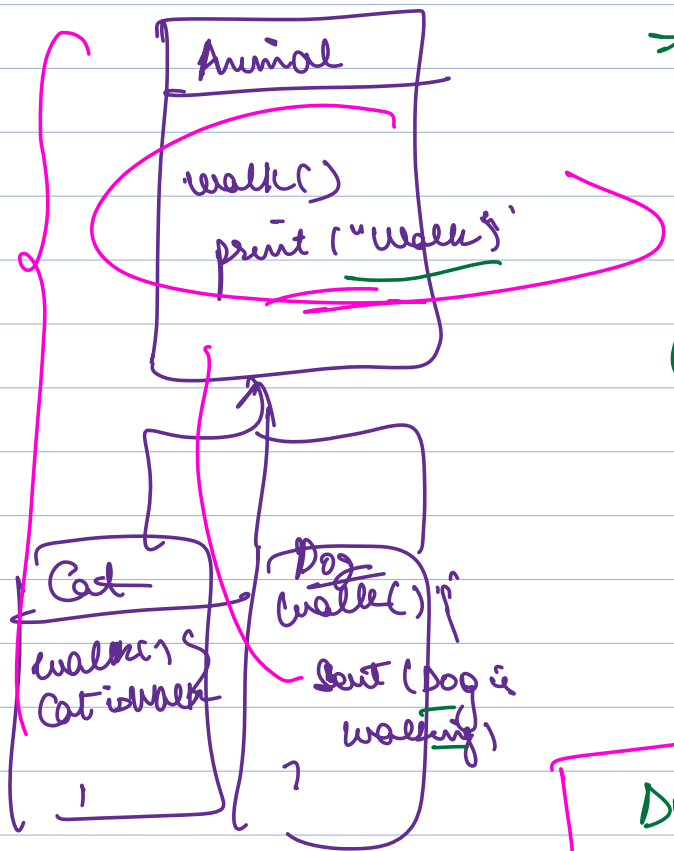
↳ replace / update / change / bypass


```
B b = new B();
b.hello();
```



⇒ child classes must have all beh of their parents

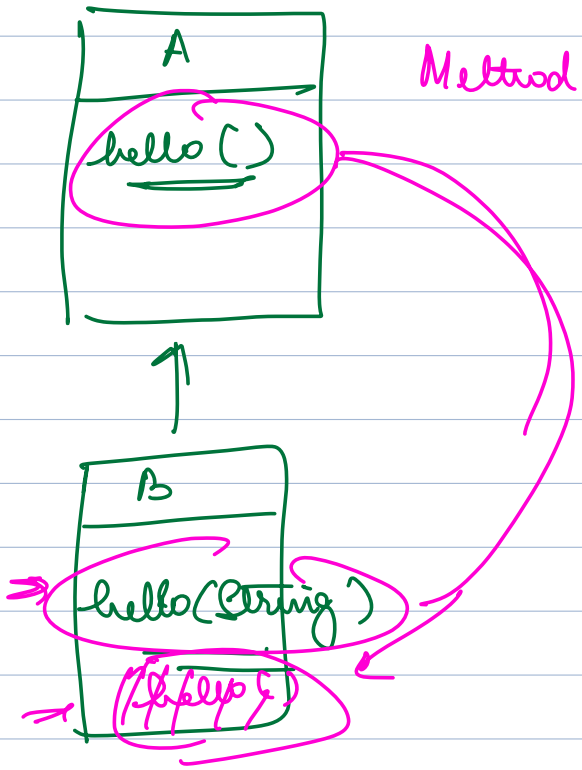
⇒ But child classes may want to perform that behaviour in a diff way



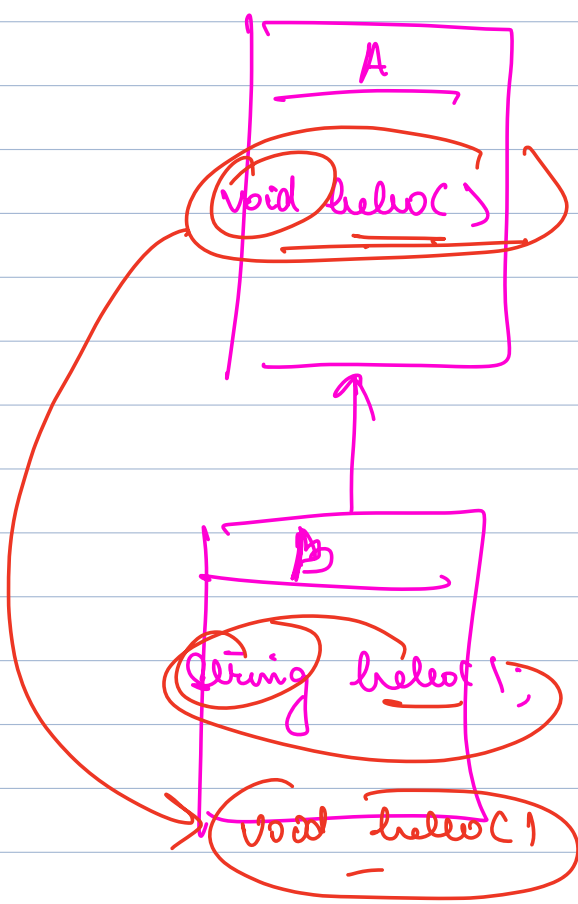
⇒ When a child class has a method which is exactly same as parent (same signature + same return type) the child's method replaces the parent's method

```
Dog d = new Dog()
d.walk()
```

Method Overriding




```
A b = new B();
b.hello("A");
b.hello();
```



Overriding ✗

Overloading ✓

Compile Time Error

```
Animal a = new Dog()
a.walk()
```

① Walk ✗

② Dog is walking. ✓

```
Animal a = new Dog()
a.walk()
```
→ Dog is walking

→ Method == Code ⇒ executed at runtime ⇒ Code / method of real Obj (not data type) will be called.

① The method that is executed is always of the obj that is actually present in the variable.

② At the compile time, compiler allows only behaviours of variable

COMPILER → Data type of variable

RUNTIME → data type of obj

```
A {
    do() {
        hello
    }
}
```

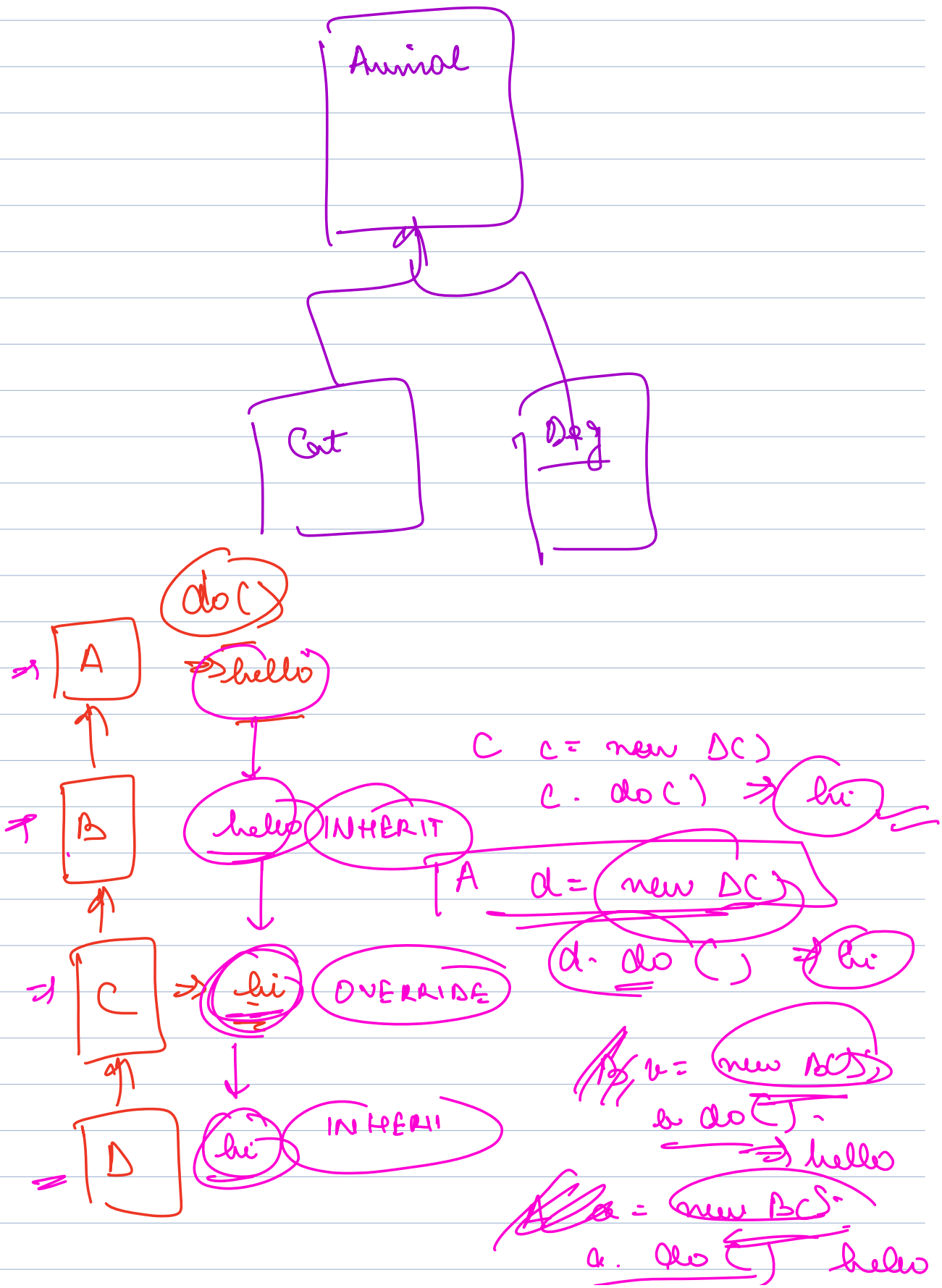
```
B extends A {
    do() {
        Bye
    }
}
```

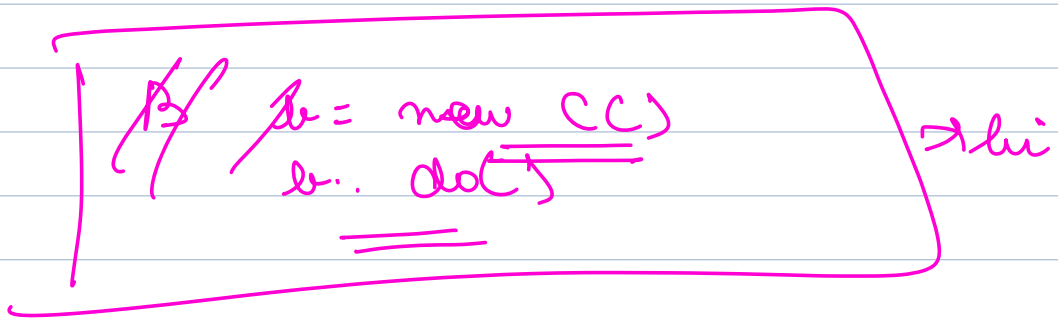
```
C extends A {
    do() {
        Hi
    }
}
D extends A {
}
```

List<A> as { new A(), new B(), new C(), new D() }

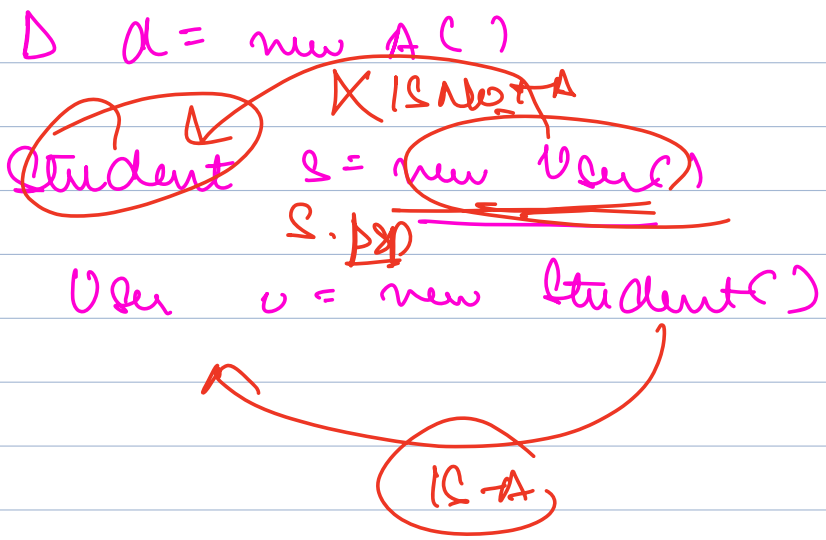
```
for(A a : as)
    a.do()
```

hello Bye Hi hello


Animal
Cat
Dog
do()
A
hello
B
hello
INHERIT
C
hi
OVERRIDE
D
hi
INHERI
C c = new D()
c.do() → hi
A d = new D()
d.do() → hi
b = new BC()
b.do() → hello
a = new BC()
a.do() → hello



⇒ after class, just try out random case



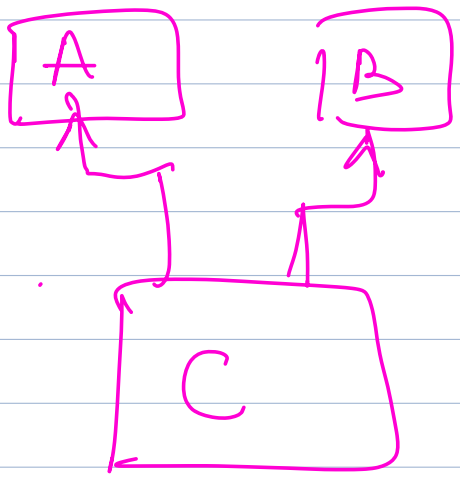
User u = new Student();
u.updateBatch()

((Student)user).updateBatch()

List l = new ArrayList()

User u = new Student

List<User> user



C extends A, B

C++ ++C

++C++ ++C