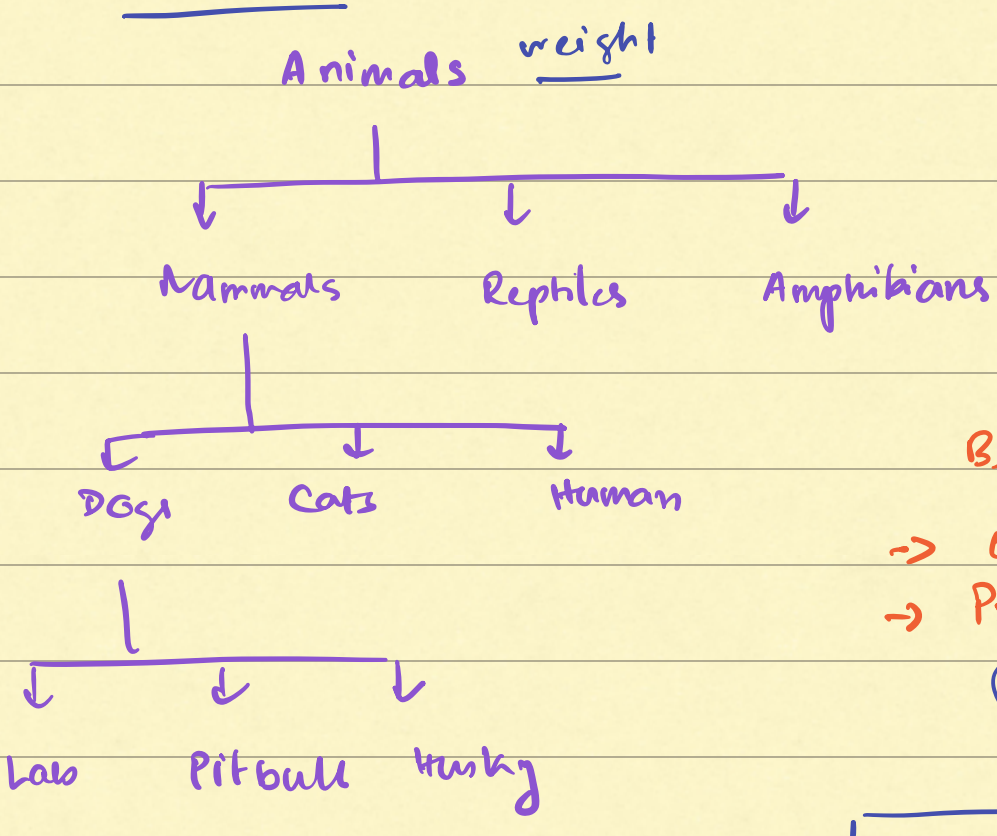
Agenda :

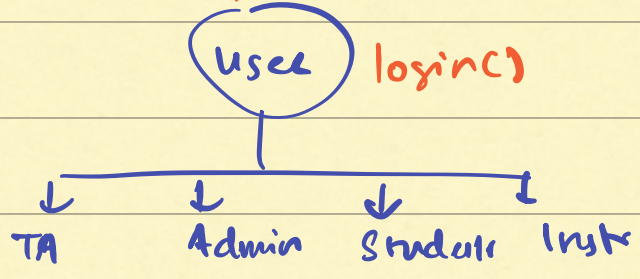
1. Inheritance
2. Polymorphism
3. 2 types of polymorphism
   - → Method Overloading
   - → Method Overriding
4. Static block & Destructors

Inheritance :



Benefits

-> Code Reusability
-> Property "



Parent

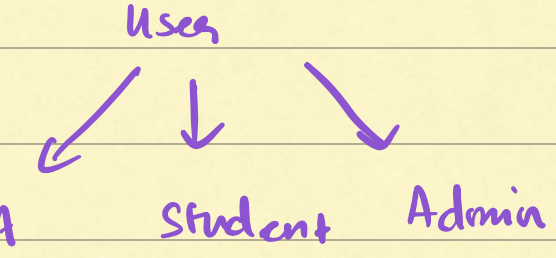
↑ set of properties & functions

child

public ✓ → Has access (Outside & inside package)
default → It can access within package.
protected → Child class from outside & inside package can access
private ✗ → Child doesn't have access to private member.

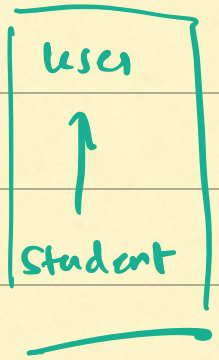
Polymorphism

many forms



TA is an user

Student is an user



void login (User u);

✓ login (Student s)

✗ login (TA t)

✗ login (Admin a)

TA t1, t2;
Student s1, s2;
Admin a1, a2;

Student s = new ...

login (s);

User u = new Student();

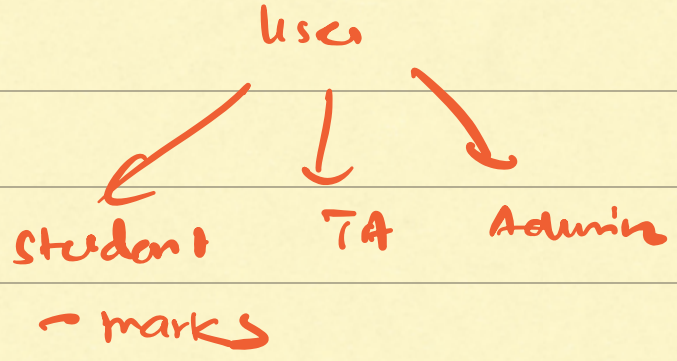
⇒ Parent class reference variable can point to child class object but not vice versa.

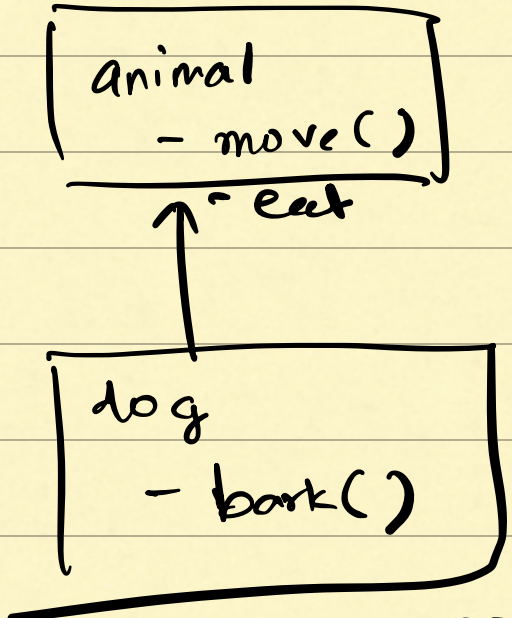
Student s = new User();

List<Admin> admins;

List<Student> students;

List<User> users;

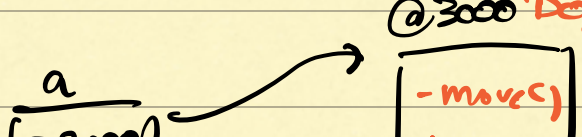




Animal a = new Dog();

a.move();

a.bark(); ✗ compile time.



@3000

- bark()

→ feed(Animal a) {

a.move();

a.eat();

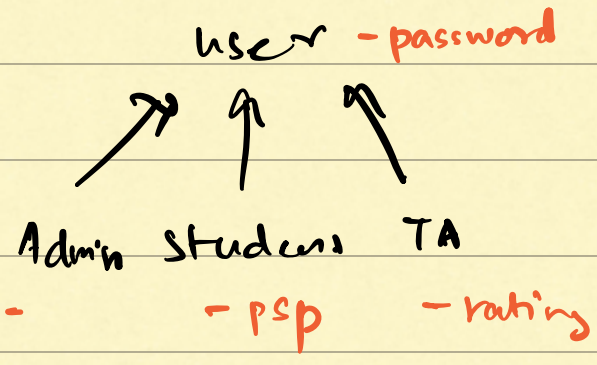
}

Dog d = new Dog();

Cat c = new Cat();

feed(d); ✓

feed(c); ✓



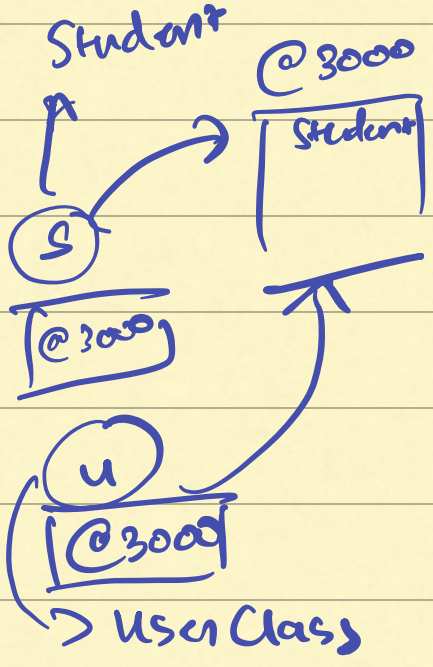
changePassword(User u) {

u.psp ✗ } ✗

u.rating ✗

u.password = "change";

}



Student s = new Student();

changePassword(s);

Admin a = new Admin();

changePassword(a);

s.psp ✓

u.psp ✗

Method Overloading (compile time polymorphism)

```
class Student {

    void Hello() {
    }

    void Hello() {
    }

    void Hello(String m) {
    }

}
```

s.hello()

s.hello("AK");

Method Signatures : (should be diff for overloading)

```
void display(String s, int y);
```

return X

display(String, int);

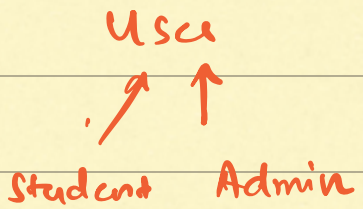
display → funcname

String, int → data types

Ex: void hello(String x, int y)

void hello(int x, String y)

✗ int hello(____)




```
Student hello() {
    returning student
}

Admin hello() {
    returns admin();
}

User u = hello();
```

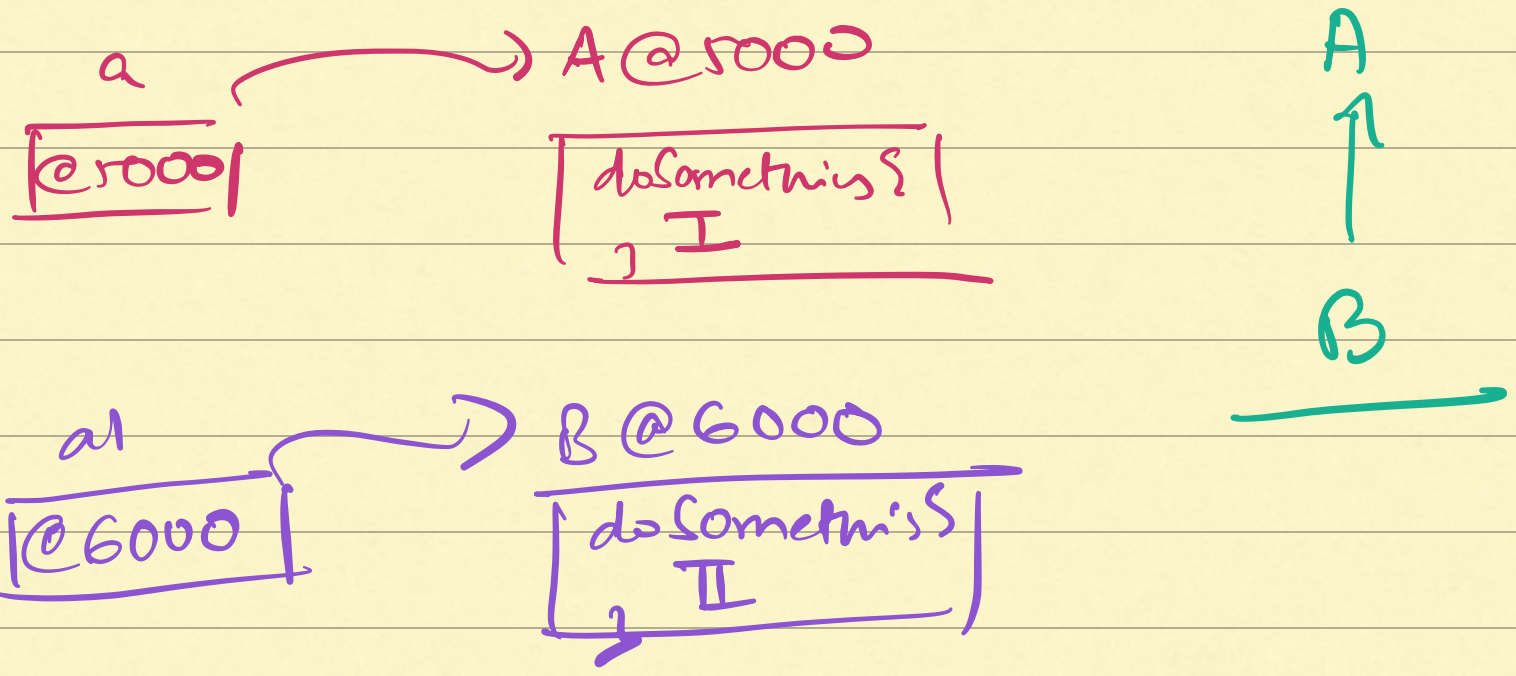
Method Overriding (Runtime polymorphism)

```
class A {
    void doSomething(String a) {

    }
}

class B extends A {
    void doSomething(String a) {      // I
    }
    void doSomething(String a) {      // II
    }
}
```

this is not written by us ↑ assuming inheritance

getting used

```
A a = new A();
a.doSomething();   // I / II
A a1 = new B();
a1.doSomething();  // I / II
```


a
@5000
A@5000
doSomething
I
A
B
a1
@6000
B@6000
doSomething
II