SOLID Design Principles ⇒ 1

Agenda

1. SRP
2. OCP
3. LSP

S
O
L
I
D

SOLID Design Principles

Software Design

rules / guidelines / values / expectations

S
O
L
I
D

⇒ 5 diff principles, if followed, will help write good code

User

Mentor

Anshi

⇒

1. Extensible
2. Maintainable
3. Reusable
4. Easy to test
5. Modular.
6. Understandability
   - Easy to read

S → Singer Responsibility

O → Open Close

L → Liskov's Substitution

I → Interface Segregation

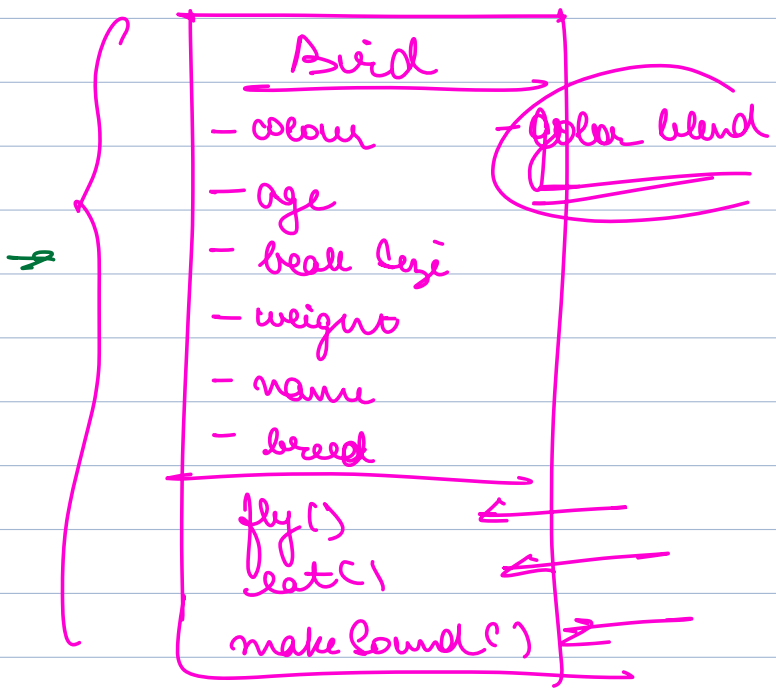
D → Dependency Inversion

Design a Bird

Amazon

Assume you have to build sfw that has to store info about all birds on planet. How well you design your class etc to do that

VO




```
Bird b = new Bird()
    b.breed = Crow
    b.name = "xyz"
    b.weight = 200
    b.age = 40

Bird b2 = new Bird()
    b2.breed = Sparrow
    b2.name = Hello
```

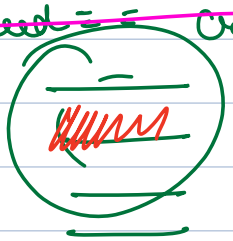
b2.weight =

b2.age = ____

b1.fly() | b2.fly()

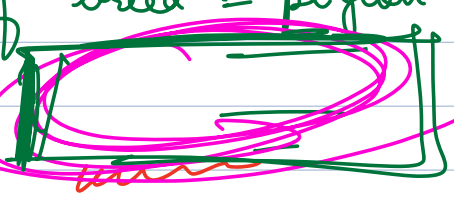
void fly() { if bird == parrot

if bird == crow :



elif bird == sparrow :

elif bird = pigeon :



elif ____

elif ____

elif ____

}

Pigeon simulator

fly()

Output

}

DAY

Dont

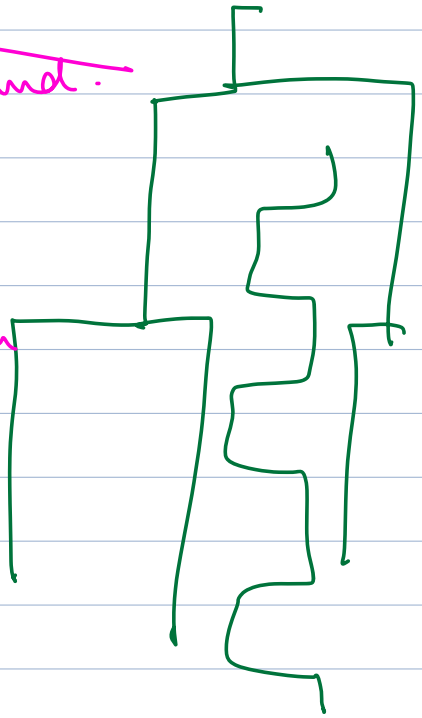
Repeat

Yourself

Problems due to many if-else

1. Difficult to understand.
2. Diff to test
3. Extensibility difficult
4. Diff for multiple people to work in parallel
5. Code Duplication and Redundancy
6. less code reuse

Regressions can happen (something working perfectly earlier stops working)



```
void fly(){}
```

```
void fly(){
  if(){
  }
  else{
  }
}
```

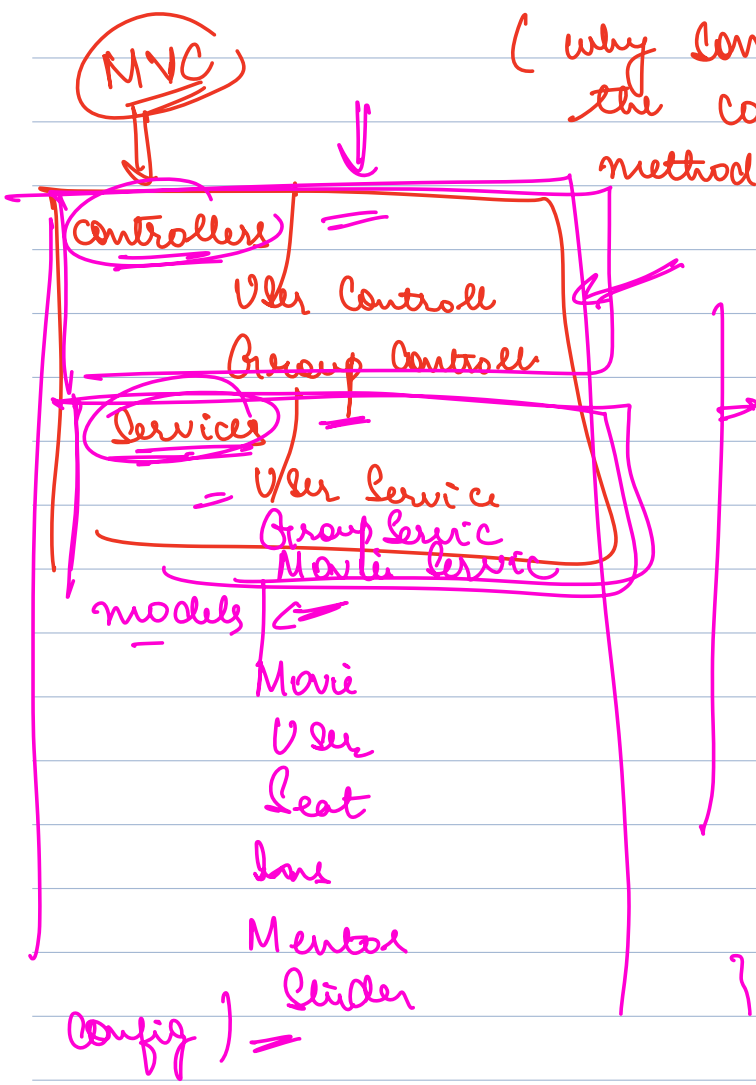
```
void fly(){
  if(){
  }
  else{
  }
}
```

(7) Violates S of SOLID -
Single Responsibility Principle

SINGLE RESPONSIBILITY PAINCIPLE

→ Every code unit (class / method / package) must have exactly well defined Responsibility

→ reason to change.
(why someone should edit the code of that class / method / package)



fly() {

⇒ To have the code on how bird will fly

}

- → Design is subjective
- → No single correct answer.
  - → every design has pros and cons
- → Don't over engineer

What was fly() of bird responsible for?

⇒ how each type of bird will fly.

~~what~~ was Bird class responsible for

⇒ all properties / methods for all types of birds

HOW TO IDENTIFY VIOLATION OF SRP

(1) Method with multiple if-else

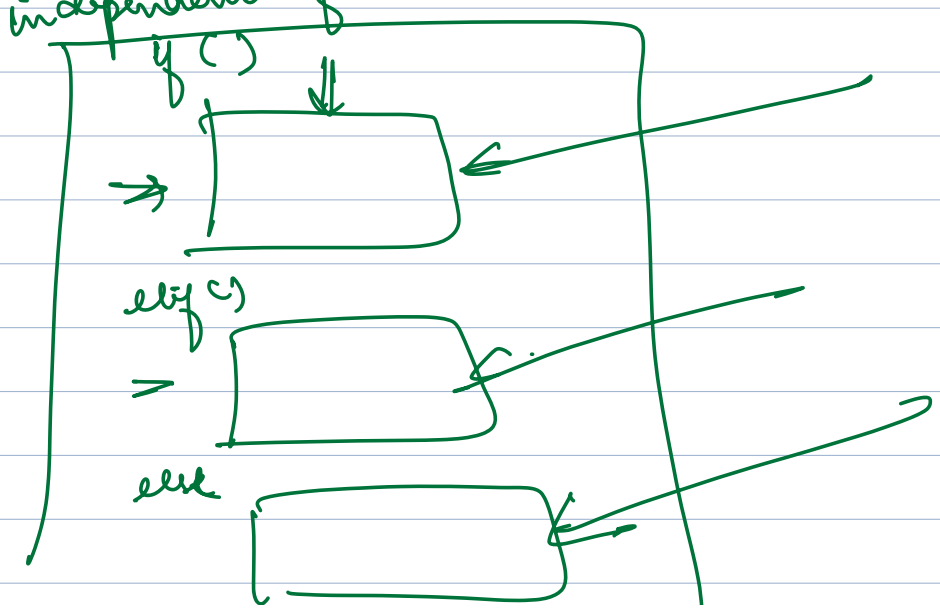
⇒ Not always bad

⇒ If multiple if-else are a part of business logic/ algo then okay

```
checkLeapYear(year) {
    if year%4 ==
        ____
        if year%100 ==
            ____
        if year%100 != year%4
            ____

    else:
        .
```

Independent if-else



```
void doSomething() {
    String inp = ____
    if (inp == 1)
        ~~SingASong()~~
    else if (inp == 2)
        Swing()
    else:
        Start()
}
```

(2) MONSTER METHOD ⇒ Big

→ Method which has code that does more than what its name asks it to do.

```
SaveToDatabase (User user, ~~Database db~~) {
    String query = "select
    ____
    ____
    ____
    ____
    ____
    ;
    Database db = new Database
    db.setURL(e)
    db.makeCon
}
```

db. execute (query).

private String createQuery() {
return ——
=
=
—
}

private DB createDBConn() {
——
=
=
}

→ Reusability.
→ Avoids Duplication

saveToDatabase() {
db String q = createSaveQuery();
Database db = createDBConn()

db. execute(q);
}

③ Commons / utils folders / files

java . util

commons / utils end up becoming a garbage place for everything you are not able to think a place of

Com . Scaler . date Utils
. time Utils
. collection

java . collection
java . datetime

④ Violates OCP of SOLID

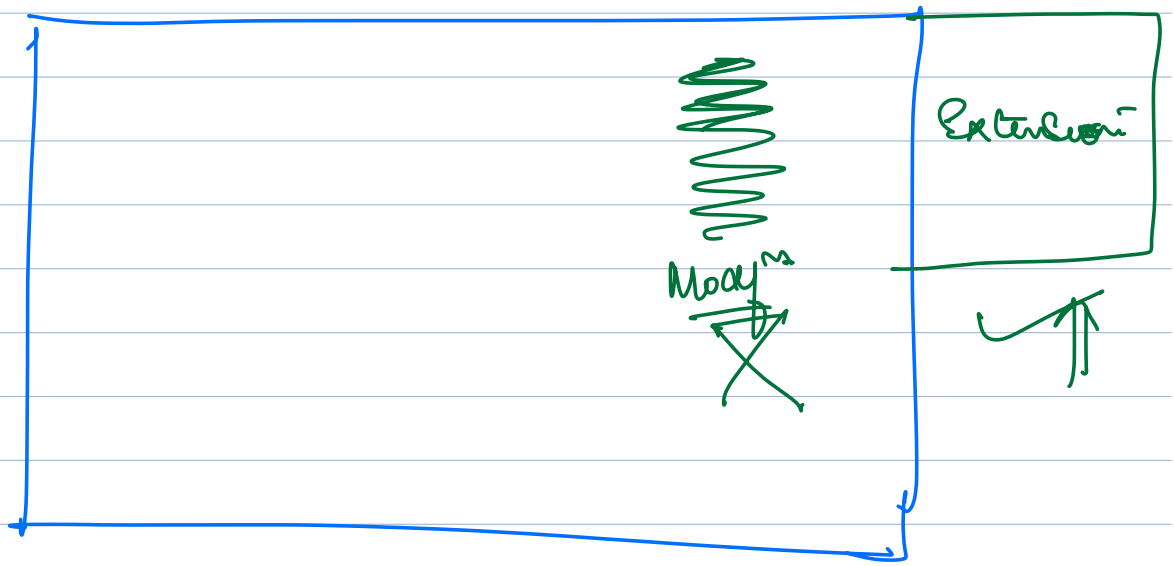
Open Close Principle

easy to add new features ?

My codebase should be open for extension but closed for modification.

adding "new features shouldn't require editing or making changes to already existing code base

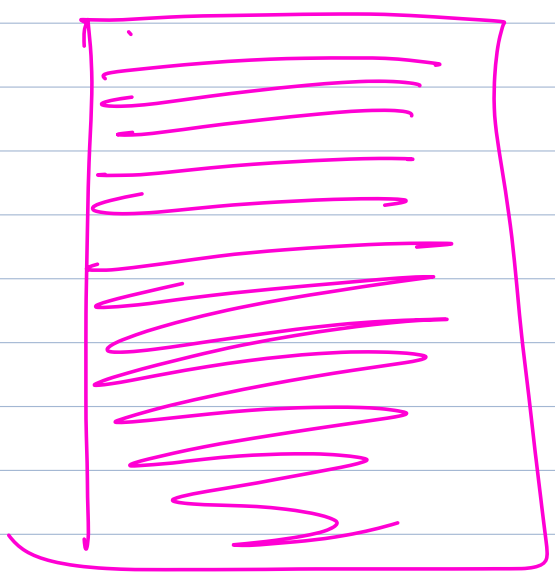
(But it should require adding new things)

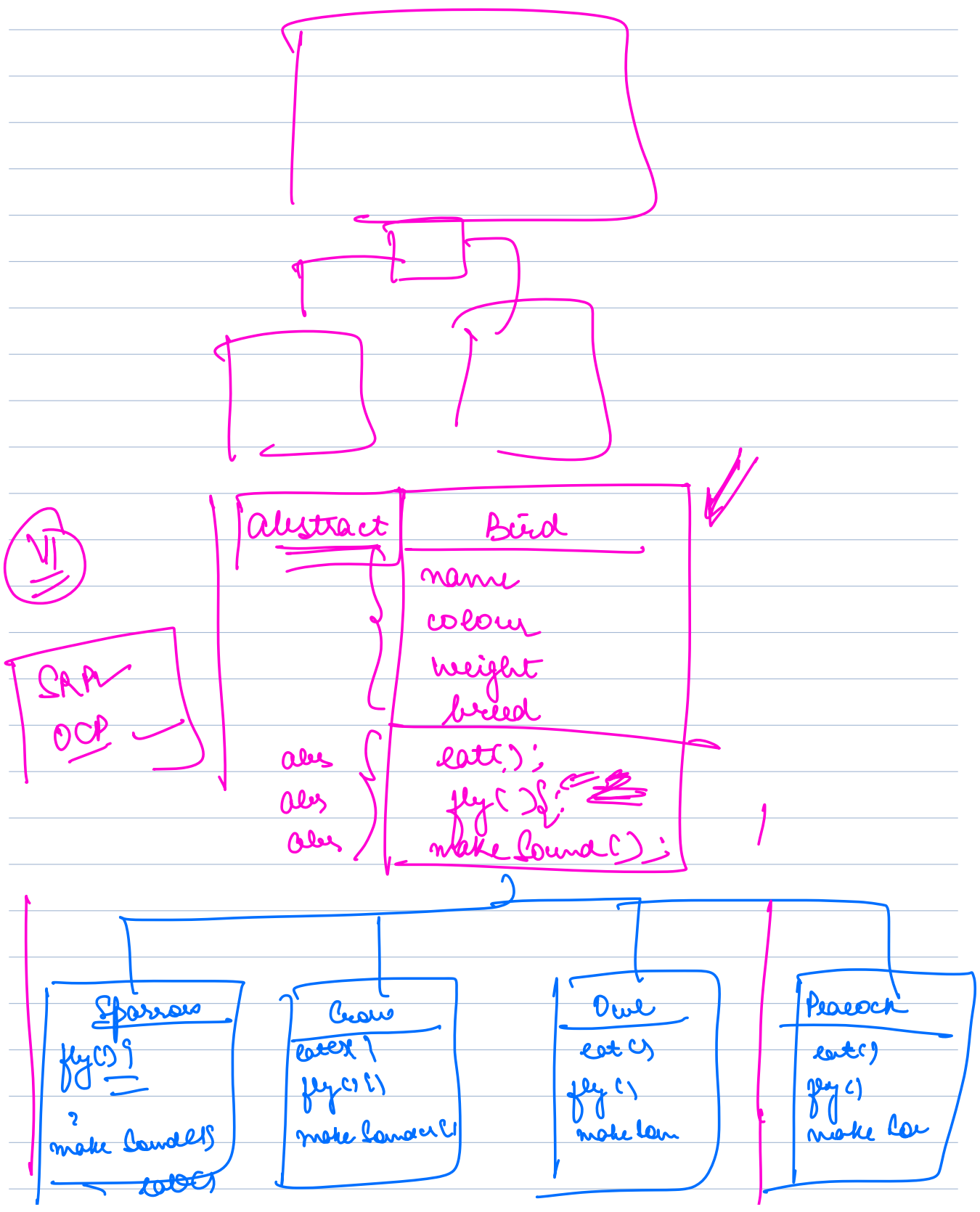


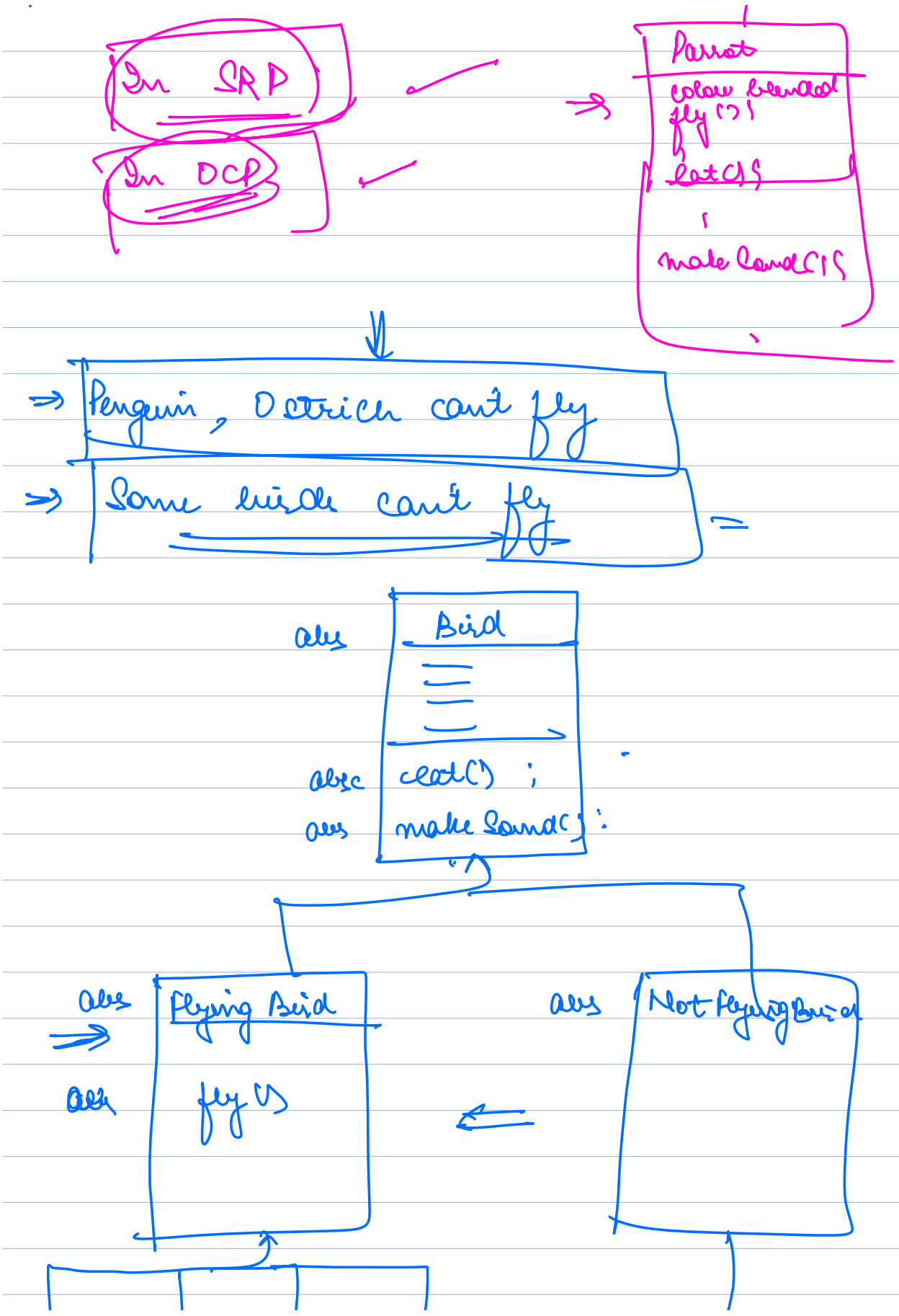
Why ?

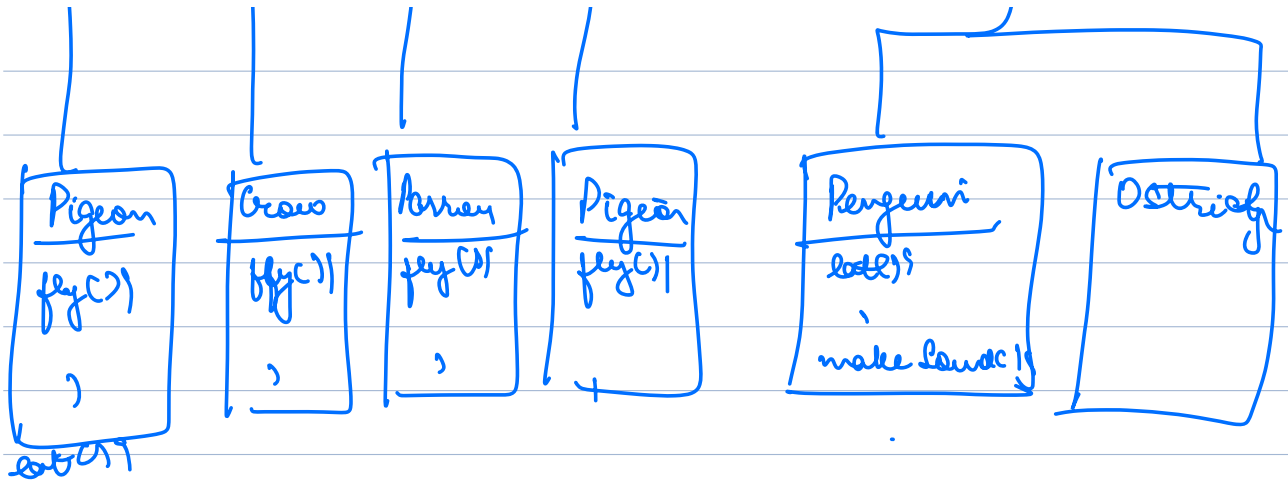
→ Regression is avoided
→ Testing is also easy
→ Merge conflicts

⇒ SRP and OCP often go hand in hand




VI
SRP
OCP
abstract
Bird
name
colour
weight
breed
abs
abs
abs
eat();
fly();
makeSound();
Sparrow
fly()
makeSound()
eat()
Crow
eat()
fly()
makeSound()
Owl
eat()
fly()
makeSou
Peacock
eat()
fly()
makeSo

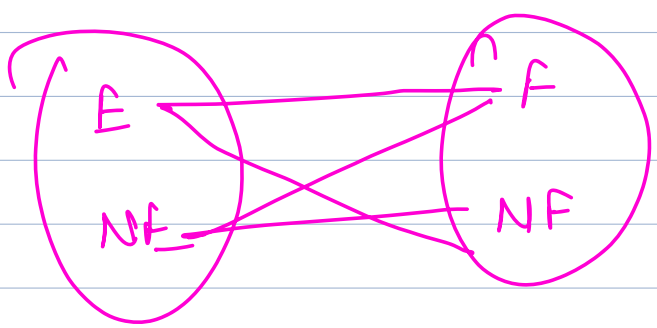

In SRP
In OCP
Parrot
colour
boundary
fly()
eat()
makeSound()
Penguin, Ostrich can't fly
Some birds can't fly
abs
Bird
abs eat();
abs makeSound();
abs
FlyingBird
abs
fly()
abs
NotFlyingBird

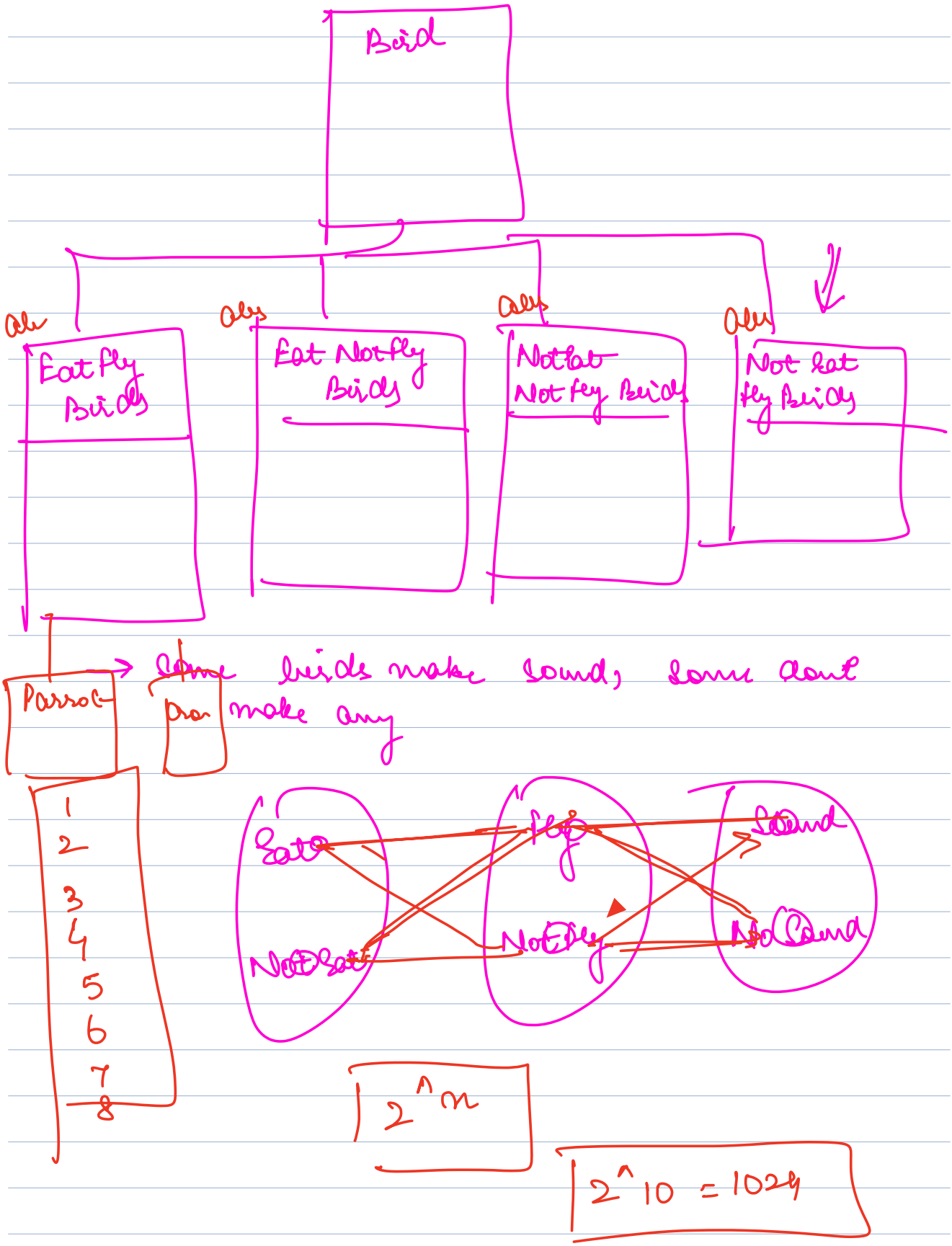


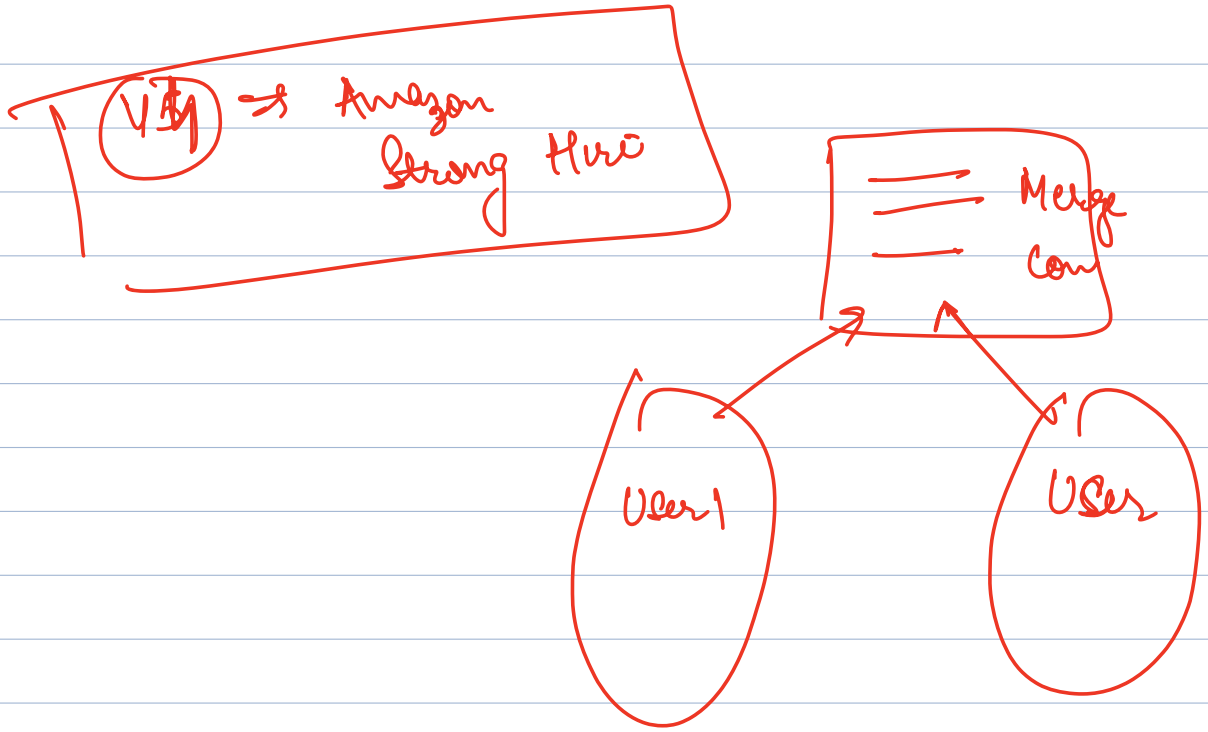
makeSound()

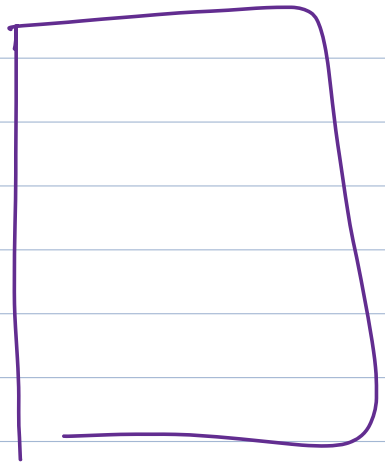
}

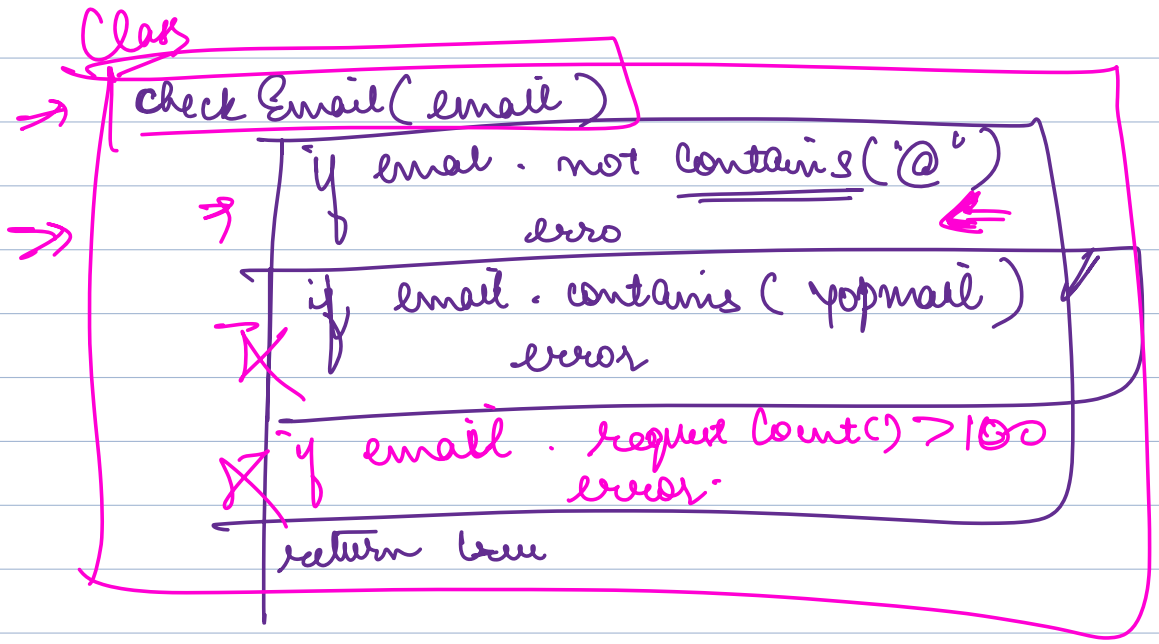
⇒ Some birds can eat, some birds can't




Bird
Eat Fly Birds
Eat Not Fly Birds
Not Eat Not Fly Birds
Not Eat Fly Birds
Parrot
Some birds make sound, some don't make any
1
2
3
4
5
6
7
8
Eat
Not Eat
Fly
Not Fly
Sound
No Sound
2^n
2^10 = 1024


VSS
Amazon
Strong Hire
Merge
Conf
User1
User2





checkEmail(email)
validate@(email);
validateDomain(email);