

# SOLID - 1

## ⊗ Agenda:-

- Single Responsibility Principle
- Open closed Principle
- Build foundation of LSP.

⊛ Single Responsibility Principle (SRP):-

① What are SOLID?

SOLID

# Design

# Principles

# Software Design

guidelines / foundation / values

- S → Single Responsibility principle (SRP)
- O → Open - closed principle (OCP)
- L → Liskov's substitution principle (LSP)
- I → Interface segregation " (ISP)
- D → Dependency Inversion " (DIP)

## What are SOLID?

↳ It's an acronym of 5 key s/w Design principles that if followed shall lead to a great software design.

A great software can be —

- ① Maintainability
- ① Extensible
- ① Readable / understandable.
- ① Easily Testable
- ① Reliable
- ① Modular
- ⋮

⑧ Designing a Bird :- (Amazon Interview Q. )

↓

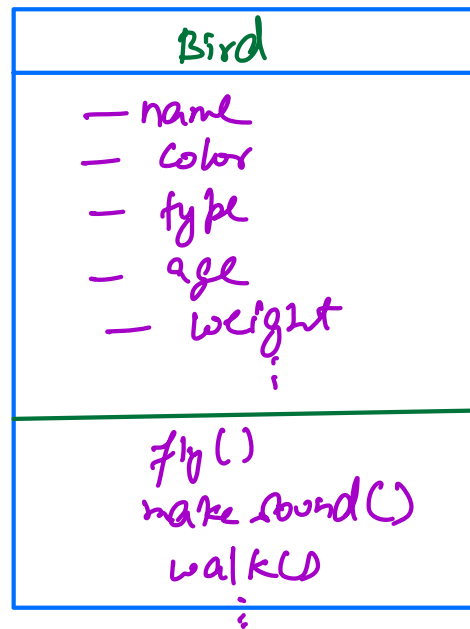
Requirement:-

Assume you are building a software in which you have to store the information about birds, so the class diagram of such software.

$V_0 \rightarrow V_1 \rightarrow \dots \rightarrow \underline{\underline{V_5}}$

VO

→ To create a class Bird with all attributes & behaviour.



Client {

    Psvm() {

        Bird Sparrow = new Bird();  
        Sparrow.name = \_\_\_\_\_;  
        Sparrow.weight = \_\_\_\_\_;  
  
        Sparrow.fly();

    }

}

```

fly() {
  if (type == 'pigeon') {
    //
  }
  else if (type == 'sparrow') {
    //
  }
  else if (type == 'owl') {
    //
  }
  ;
}

```



Difficult to maintain

⑨ what are problems with too-many if-else?

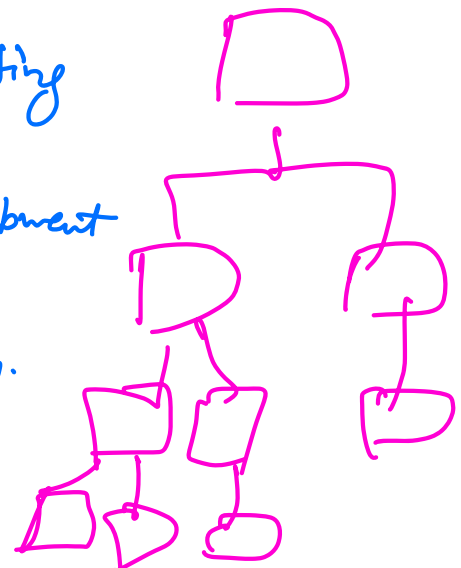
→ ① Understandability.

② Readability.

③ Difficult to Testing

④ concurrent Development complexities.

⑤ Code Reusability.



① violate 2 of SOLID

① violates DRY principle.

↓  
(Don't Repeat Yourself)


⑦ SRP :-

SINGLE RESPONSIBILITY PRINCIPLE

↳ Every code unit (method/class/package) in our codebase should have exactly 1 responsibility.



→ exactly one reason due to which we might want to change that code unit.



- $\Rightarrow fly() \rightarrow SRP$  X  
 $Bird \rightarrow \text{X } \underline{\underline{(SRP)}}$

```
① DB db = new DB(  ).connect();
```

Overthinking!

② LID is very Subjective!

③ If you are able to understand the tradeoff, of your design, then you're good.

④ How to identify violators of SRP:-  
→ 3 common places.

① Method having multiple if-else:-

```
{  
  if (age > 18)  
    _____  
  else  
    _____  
}
```

```
{  
  if n == 10  
    x _____  
  n == 11  
    x _____  
  n == 12  
    _____  
}
```



↳ Business logic → if-else  
(multiple)

↓

Okay:

↓

Doesn't violates  
SRP.!

(2.)

Monster Method :-

↓  
huge!

↳ if it doing multiple things.

↳ it's a method that has a piece of code which does a lot more than what it's name expects to.

saveToDatabase ( user, Database db ) {

generating  
a  
query

String query = "  
+ "  
+ " user.email "  
+ "

creating  
a  
connection

db.getConnection();  
db.connect();

saving.

{  
db.save ( user );  
}

generateQuery ( ) {

}

createConnection ( ) {

}

saveToDB ( ) {

}

save To DB ( ) {

generate Query ();

create conn ();

db.save (user);

}

Responsibility  $\Rightarrow$  how to do something

3.

Commons / utils Package :-

$\rightarrow$  Java

$\rightarrow$  discouraged.

$\rightarrow$  end up into becoming a huge package,  $\Rightarrow$  SRP ~~X~~

Summary:-

① SRP → every code unit  
⇒ exactly 1  
responsibility.

→ How to identify:-

- ↳ multiple if-else
- ↳ Monster
- ↳ Utils/Commons

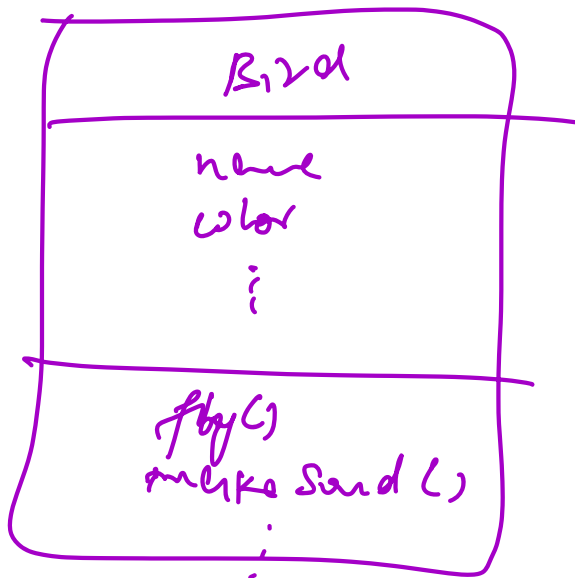
Break fill 08:20 AM IST

## (\*) Open - Closed Principle:-

↳ A codebase should be  
OPEN for extension,  
CLOSED for modification

↳ My codebase should make  
it easy to add new  
features.

But adding new features  
should not require  
making much changes to  
the already existing codebase.



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

New Feature



Subtract a new bird  
— Peacock!

why OCP?

if we modify a code?  
↓

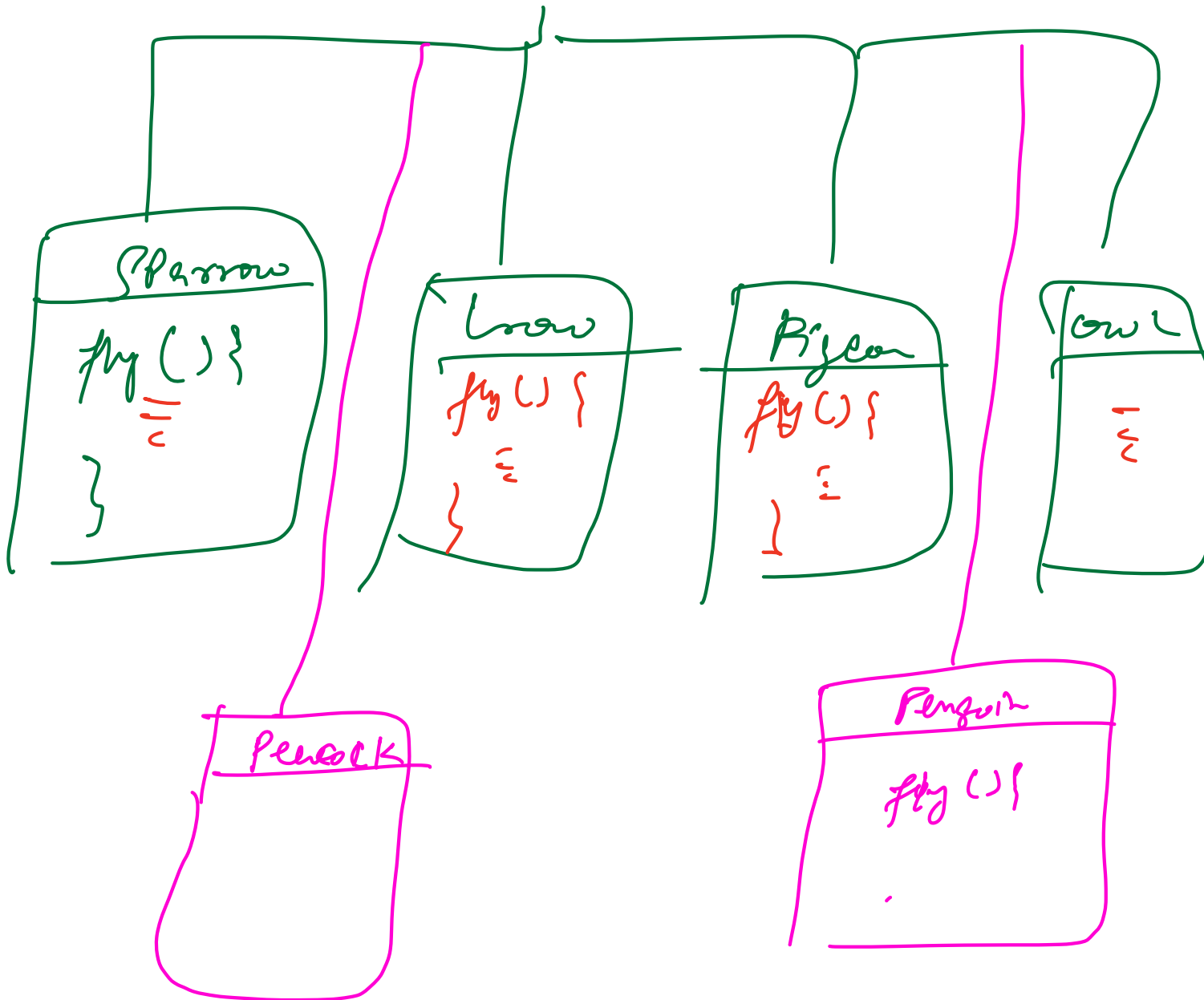
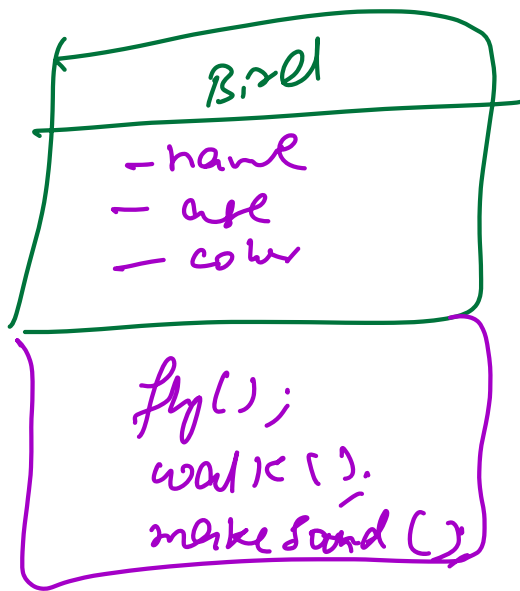
① Some changes that were  
previously working  $\Rightarrow$  might  
break up.

REGRESSION

② Testing  $\Rightarrow$  Modify existing  
Test cases!

Bird Vo  $\rightarrow$  SRP ✗  
OCP ✗

Version 1



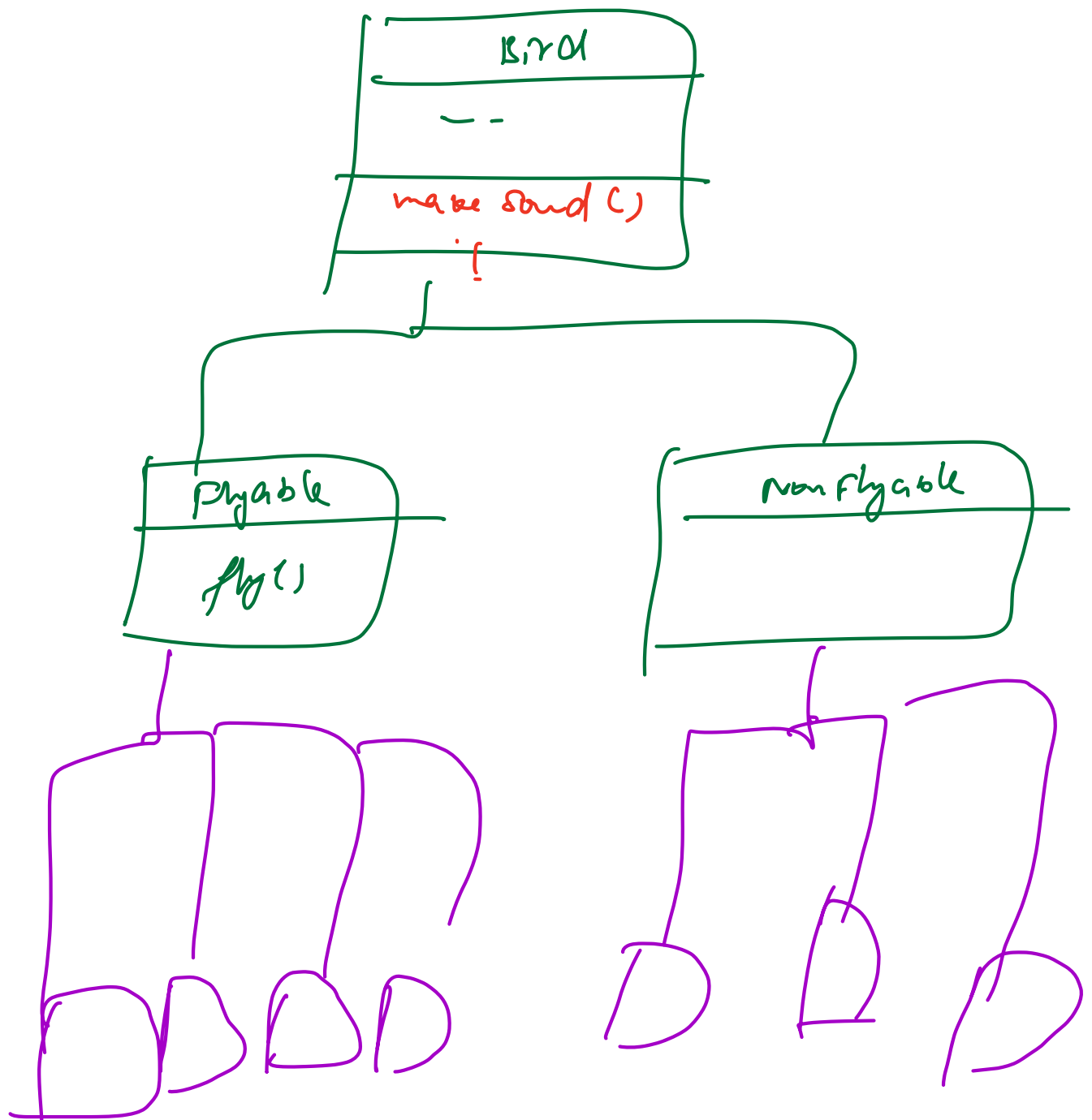
OCP ✓  
SRP ✓

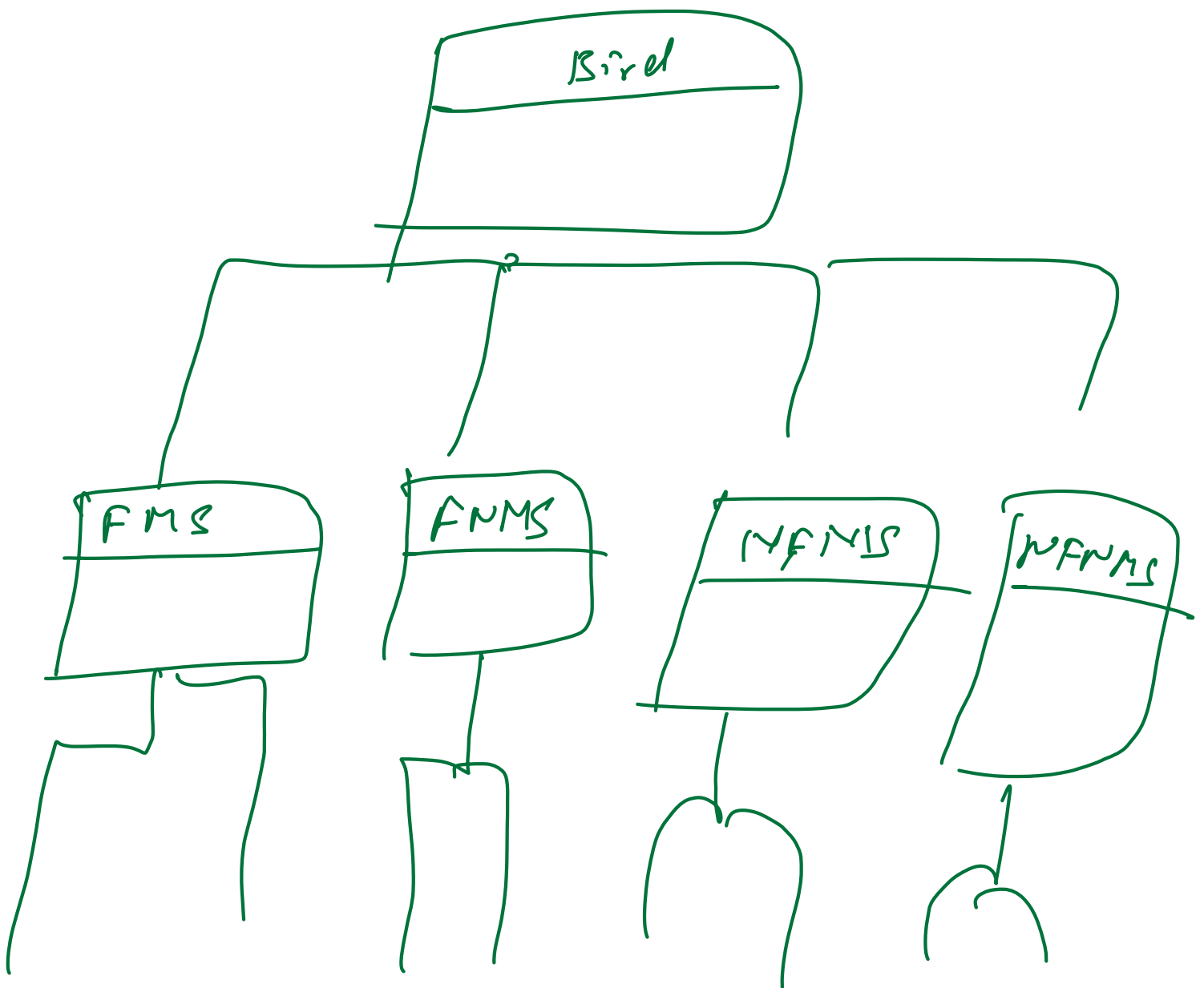
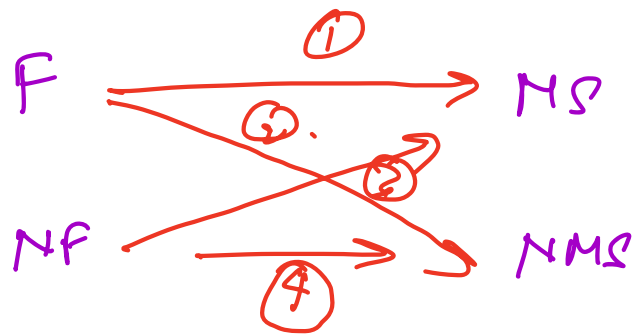
Penguin → They cannot fly!

- ① Not implement fly() in Penguin.
- ② Throw exception fly() in Penguin

③ Create 2 child - Flyable  
- Non-flyable







$$\underline{\underline{10}}$$

$$\underline{\underline{2^{10}}} = \underline{\underline{1024}} \text{ classes.}$$

Class Explosion!

$$\underline{\underline{n}} \rightarrow 2^n$$

$$v_0 \rightarrow v_1 \rightarrow \dots \rightarrow \underline{\underline{v_5}}$$