Backend LLD 2

10 classes.

- 2 classes → SOLID → Build
- 7 classes → Design Patterns.
- 1 class → UML diagrams

+ contest : attempt

+ mock : Backend LLD 1 content

Skill

Reattempt of prev module must be live ✓

Agenda

SOLID → next

principle

↳ guidelines / fundamentals.

better software design

Extensible

Maintainable

Readable

Reusable

Testable

Modular.

S Single Responsibility principle

O Open/closed principle

L Liskov's substitution Principle

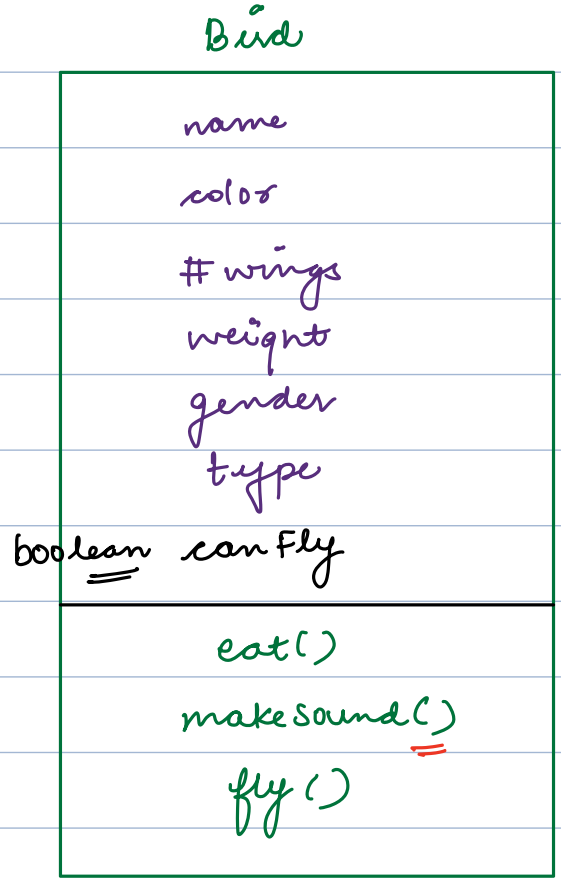
I Interface Seggregation

D Dependency Inversion.

Design a Bird

design a software system which store the attributes & behaviour of diff birds.

V0




```
Bird b1 = new Bird()
    b1.type = "pegion"
    b1.name = "poopoo"

Bird b2 = new Bird()
    b2.type = "sparrow"
    b2.name = "spi"
```

b1.makeSound() b2.makeSound()

Sounds are not same.

```
makeSound() {

    if ( type == "pegion" )
        ==
    else if( type == "sparow" )
        =
    else if (          )
        =
    :
}
```

It violates (S) of SOLID

Single Responsibility Principle

class / method / package

Every code unit should have 1 responsibility to take care of

OR

There should be single reason to change the code.

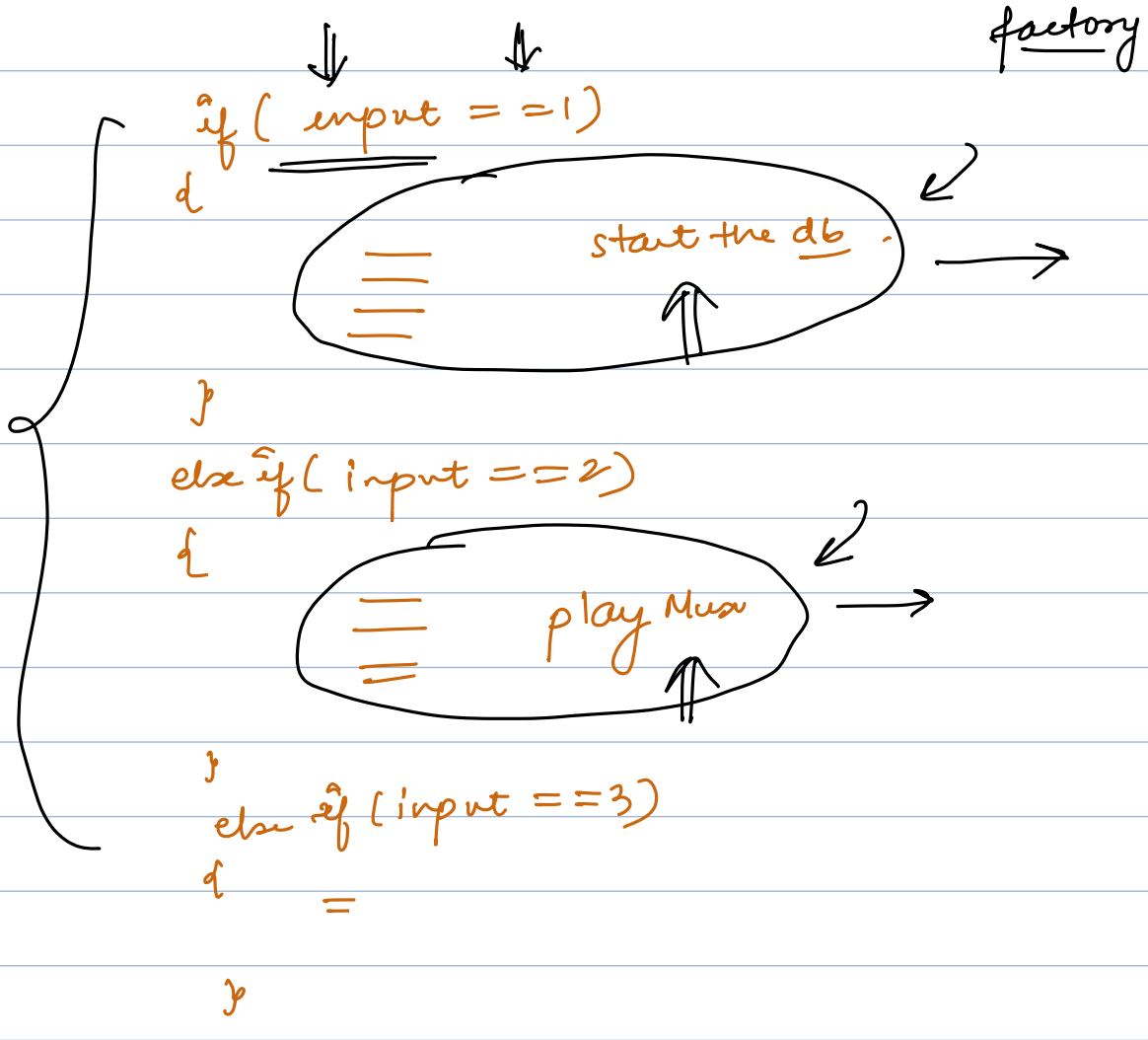
makeSound() X

Bird X

How to figure out violation of SRP :-

① Method that has a lot of if-else

Business logic — checkLeapYear

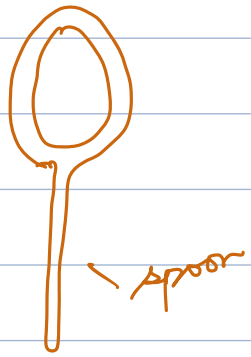


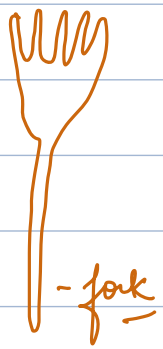
(2) Monster method

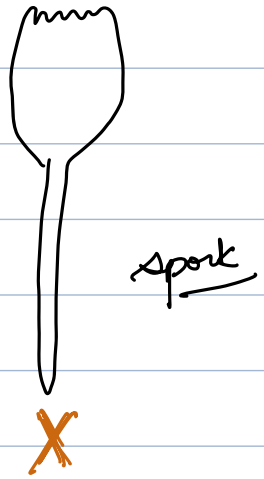
when a method is doing more than what it is supposed to be doing

```
saveUserTodb(User usr){
  ① { query = "_____";
  ② { start the db conn
  ③ { db.execute(query);
}
```

new DataBase()
db.start()
db.connect()







③ utility

UserUtility ←
UserUtility

OCP: Open closed Principle

open for extensions but closed for modifications.

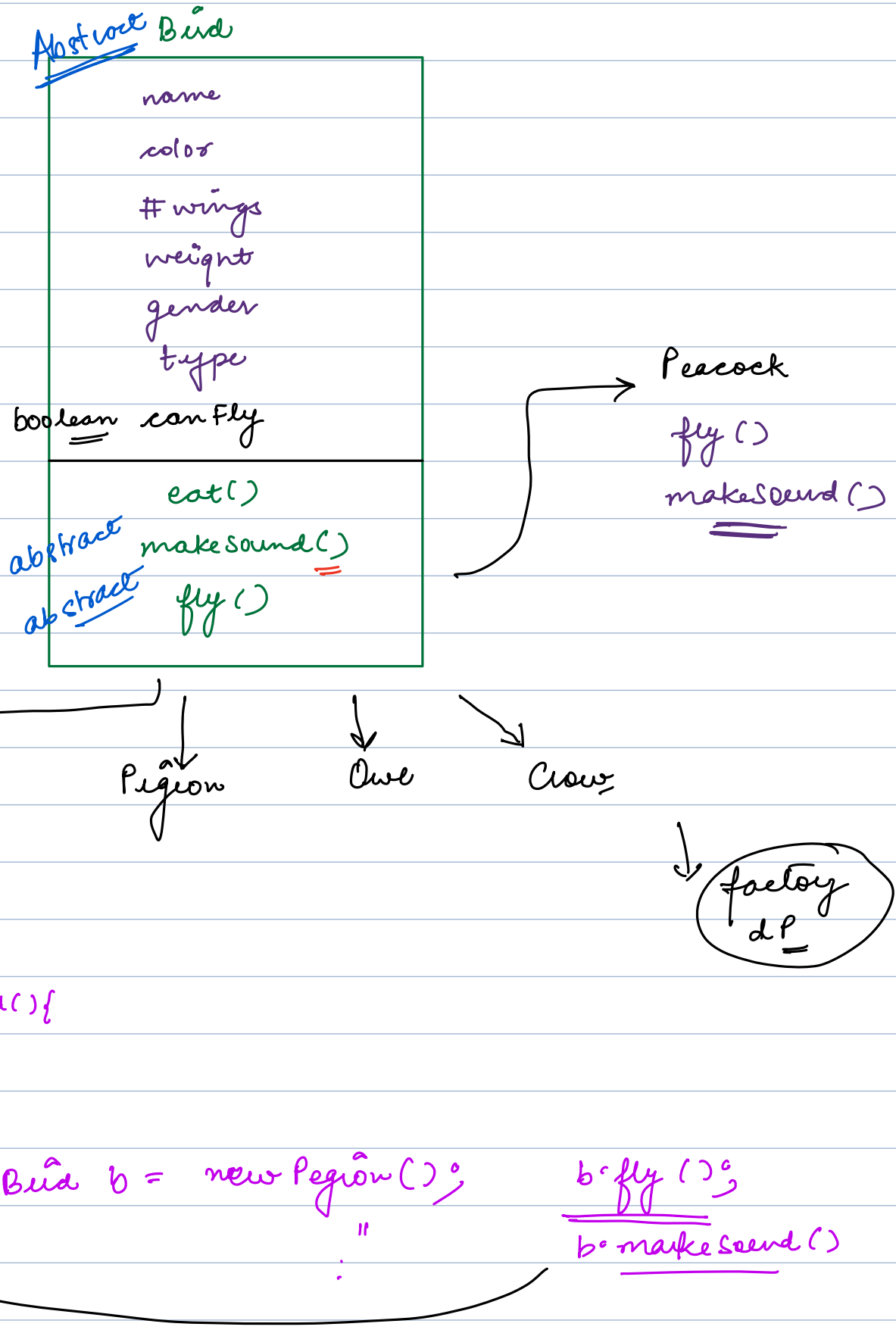
- easy to add new features
- should not be a lot of code change in existing

```
makeSound() {
    if ( )
    {
        ≡
    }
    else if ( )
    {
        ≡
    }
    else if ( )
    :
    }
    if else ( ) {
        ≡
    }
}
```

Break:
10:33 pm

Bird class should be more generic, not specific.

(V1)



Penguin

↳ can't fly but can swim.

```
Penguin extends Bird {
    fly() {
    }
    makeSound() {
    }
}
```

→ ① leaves this empty
② throw exception

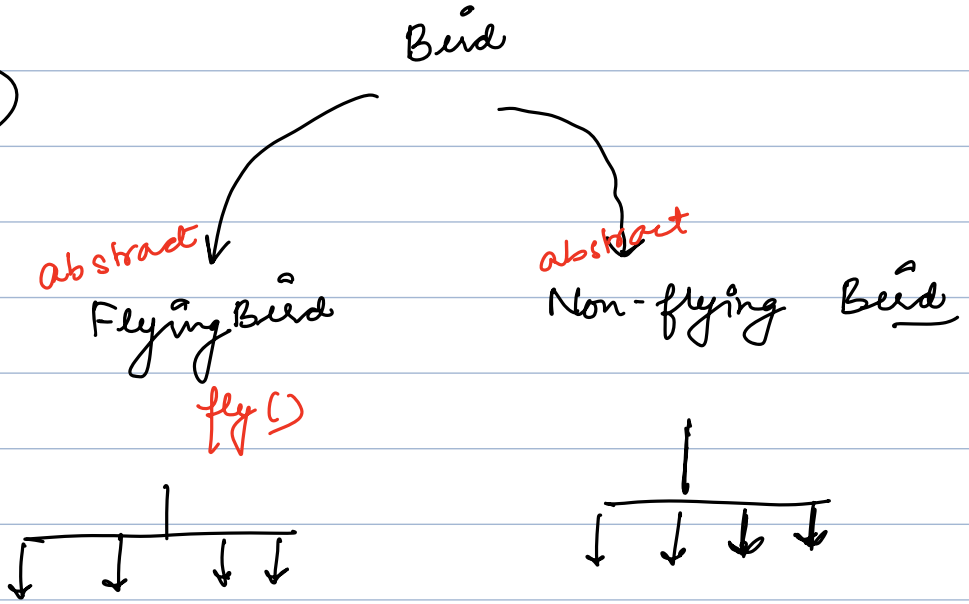
```
Bird b = new Penguin();
b.fly();
```

surprise

If an entity doesn't support a method it should not have a method.

Some birds can fly and some can't.

V2



Fly + sound | Fly + NS | NF + Sound

NF + NS

✗ | n variables ⇒ 2^n classes | : class explosion.

10 MCQ's

User { isVerified / isBlocked }

enum ⇒ { VERIFIED, BLOCKED, NOTVERIFIED }