Backend LLD & dev 2

↓

design principles & patterns.

↓

10 classes.

2 — 7 — 1

Solid → MCQ's

patterns → Code → Singleton → 3 problems

UML diagrams → MCQ's

- Retention
- PSP
- contest

{ Head First design patterns.

{ refactoring.guru

Agenda

SOLID

design principle → ?

rule

guidelines / fundamentals

better software design

Modular

Reusable

Maintainable

Extensibility

Readable

S : single Responsibility

O : Open Closed Principle

L : liskov's substitution

I : Interface seggregation

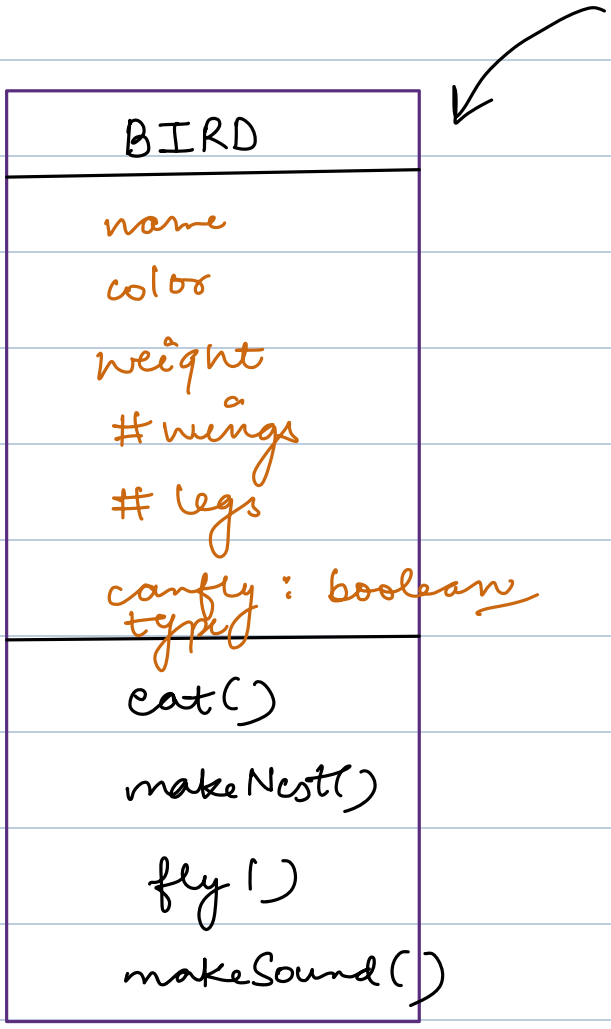
D : Depency Inversion

Injection

Design a Bird

design a software system which stores attributes & behaviours of diff birds.

V0




```
Bird b1 = new Bird();
b1.type = "peacock";
b1.makeSound();

Bird b2 = new Bird();
b2.type = "sparrow";
b2.makeSound();
```

```
makeSound() {
    if ( type == "peacock" )
    {
        ==
    }
    else if ( type == "sparrow" ){
        ==
    }
    :
}
```

makeSound
Bird

→ package, class, methods.

Every code unit should have a single responsibility to take care of

OR

there should be a single reason for any code unit to change

— Methods — lot of if-else.

Buisness logic — checkLeapYear

```
doSome(input)
{
    if (input == 1)
    {
        ...
    }
    else if (input == 2)
    {
        ...
    }
}
```

start the db con

play the music

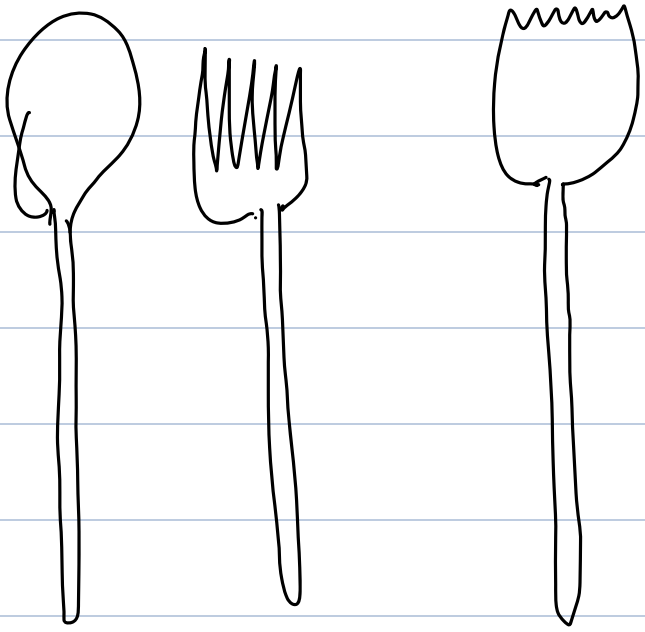
(2) monster method

Observer

process checkout ( ) {

- validate the shopping cart
- apply discounts
- calculate taxes & shipping fees
- payment
- order confirmation
- Seller
- Buyer
- Inventory.

}



Utility folder } → DustBin

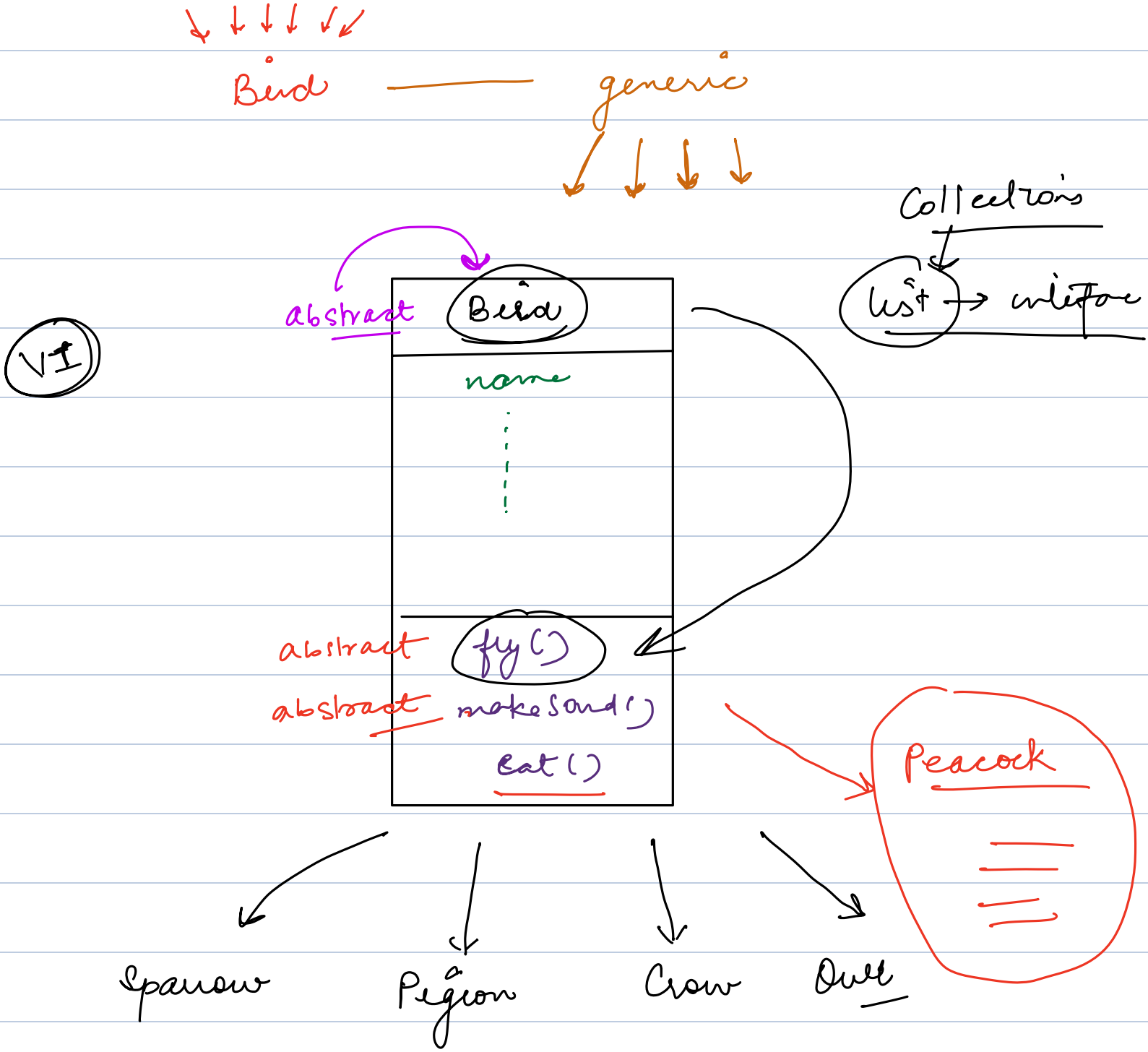
Student Utility

⇒ { userHelper
UserUtility
userUtility

Open closed Principle

open for extensions but closed for modifications

- open for extensions: easy to add new features.
- closed for modifications: should not change a lot of existing code for new features.




```
Bird b = new Peacock();
b.makeSound();
```

Support Penguin → can't fly

↓

create a new class.

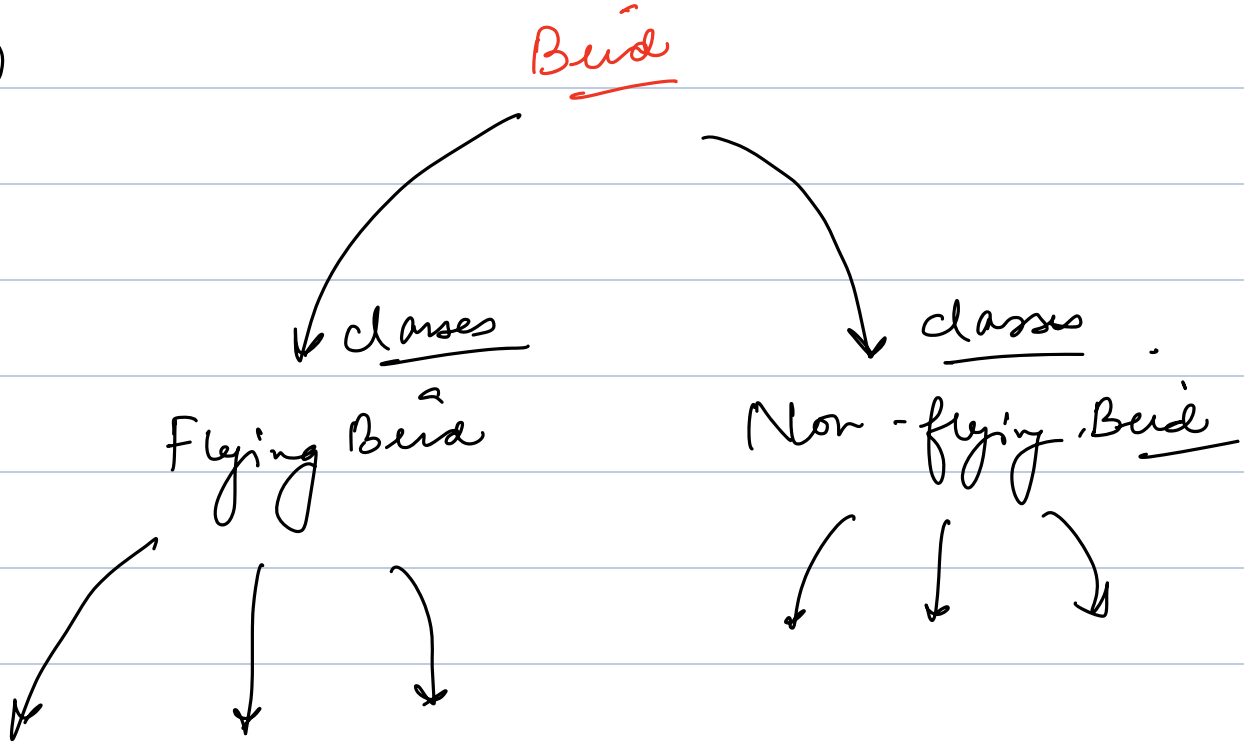
```
Penguin extends Bird {

    fly() {
        // (1) leave it empty
        // (2) Throw exception
        // (3) "I can't fly."
    }

}
```

```
doSomething(List<Bird> birds) {

    birds.get(index).fly();

}
```

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

Some Birds can fly & some can't.

(V2)

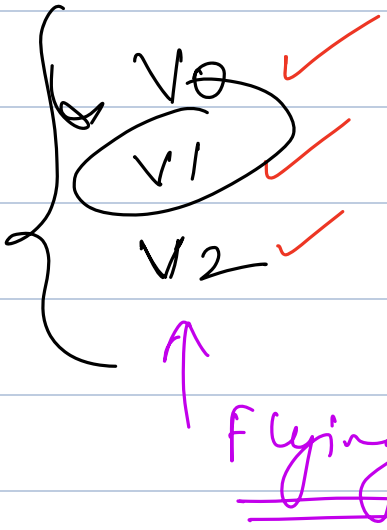


Fly ✓ X — 2
Swim ✓ X — 2
Sound ✓ X — 2

2^n → 8 classes

↓

class explosion



SRP

OCP

Flying Bird