Backend LLD 2

$$\begin{tikzcd}
 & \text{Backend LLD 2} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{SOLID} & \text{Design Patterns} & \text{UML Diagram}
\end{tikzcd}$$

SOLID
2 classes

Design Patterns
7 classes
10 DP
(23-24 DP)

UML Diagram
1 class.

<u>contest</u> + <u>mock</u>

<u>Agenda</u> :

S - Single Responsibility Principle
O - Open Close Principle
L - Liskov's Substitution "
I - Interface Segregation "
D - Dependency Inversion "

SOLID Principles

↳ Guidelines / fundamental foundation of your decision

SOLID helps us to make software more :

- Extensible
- Maintainable
- Readable
- Understandable
- Testable
- Modular
- Reusable.

1) Design a Pen
2) Design a Bird

} Google, Walmart Adobe.

Design a Bird.

V0

```
Class Bird
-----------
 - color
 - weight
 - # wings
 - type
 - name
-----------
 - fly()
 - makeSound()
 - eat()
```

```
Bird b1 = new Bird();
b1.type = "Sparrow";
b1.name = "Tweety";

Bird b2 = new Bird();
b2.type = "Pigeon";
b2.name = "Raja";
```

Pigeon
- makeSound()

Crow
- makeSound()

```
- makeSound() {

    if ( type == 'crow'){
        // crow sound code
    } elseif (type == 'Pigeon'){
        // Pigon sound code
    }
    :
```

:
}

Single Responsibility Principle:

- Every code unit should be responsible for a single behaviour

- There should be a single reason to change a code unit.

code unit – class / function / package.

function MakeSound : Violating SRP
class Bird : violating SRP

```
makeSound()     | fly()
{               |   if (  )
  if ( )        |   else
  else ( )      |
                |
}
```

Use case of Designing Bird :

Video Game	Bird Selling

How to identify SRP violations:

- Methods with multiple if-else:

main method -

```
if (input == 1) {
    take Name
} if (input == 2) {
    print Name
} if (input == 3) {
    update Name
  }
}
```

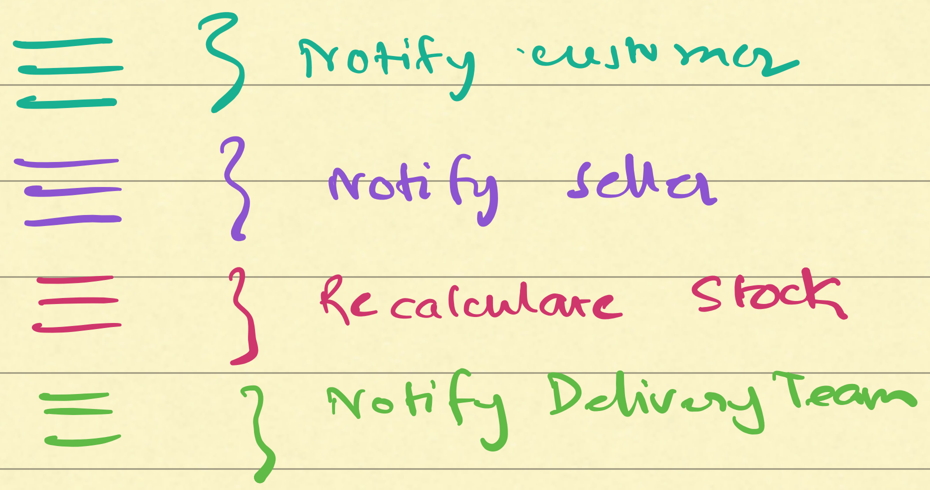
leap-year code

```
if (   )
else if (   )
else
```

- Abstract factory

- Monster Methods:
    - Complex & excessive lines of code.
    - It does more than the name suggests

```
afterOrderPlaced {
```



```
}
```

↓

```
afterOrderPlaced {
    notify Customer ( );
    notify Seller ( );
      :
}
```

3. <u>utility class / method</u>

<u>Break Till : 8:20</u>

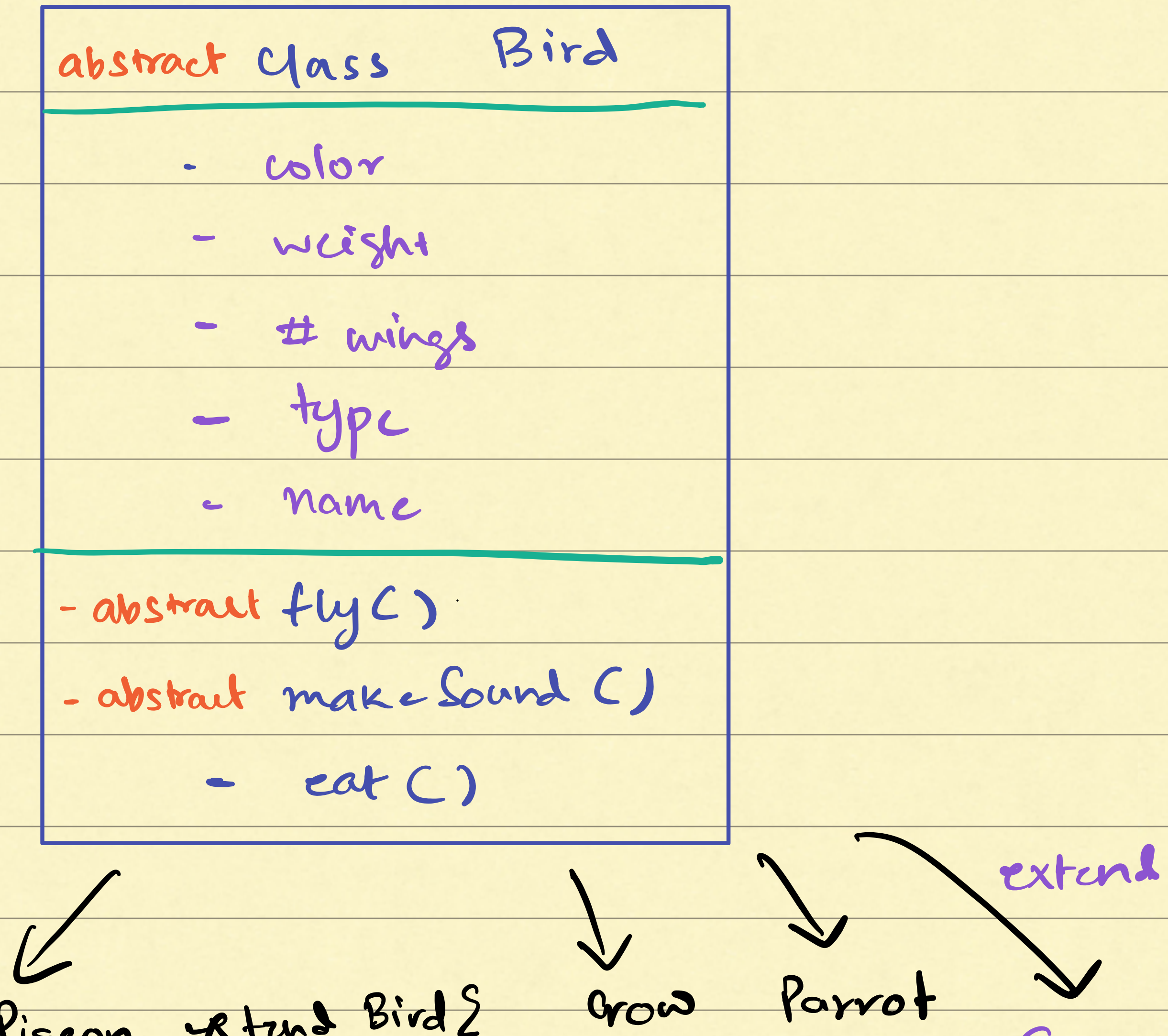
2. Open Closed Principle

- Open for extension, but closed for modification

↳ when we are adding new features

```
makeSound()                                  fly() {
  if(---){                                   if(   ){
     ≡                                           ≡
  else if(-) {                               else if(){
     =                                           ≡
  }else if(type=="parrot"){ }
     ≡
  }
}
```

(vi)



```
class Pigeon extend Bird{
    fly() {
        ==
    }
    makeSound() {

    }
}
```

```
Sparrow
{   fly() {
        ==
    }
    makeSound() {
        ==
} }
```

```
Bird b1 = new Pigeon();
    b1.name = "xyz";
```

$$\begin{tikzcd}
 & & \text{abstract Bird} \arrow[dll] \arrow[dl] \arrow[d] \arrow[rr] & & \text{Penguin} \\
\text{Pigeon} & \text{Crow} & \text{Sparrow} & &
\end{tikzcd}$$

Pigeon
- fly()
- makeSound()

Penguin
- fly()
- makeSound()

```
class Penguin extends Bird {
    void fly() {

        throw new birdCantFly();
    }
}
```

→ ① leave it empty

② Throw Exception

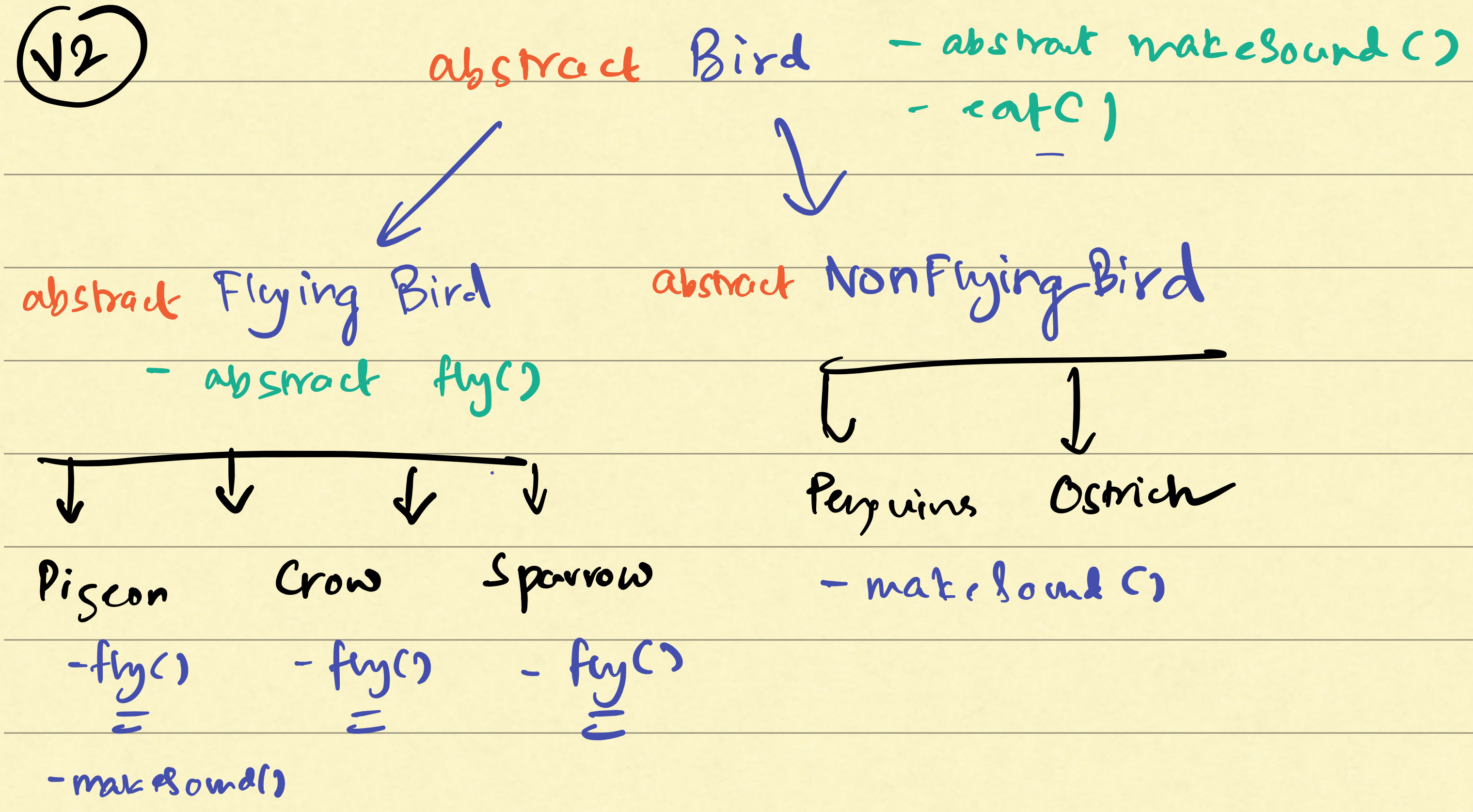
```
function( Bird b ) {
    b.fly();
}
```

← new Pigeon(),
new Penguin().

b.fly(); ↳ throws an exception.

A class should have a method implement if we are able to call it.

V2



Swimmable Birds

NestMaker Bird

$\boxed{2^3}$

F | NF | S | NS | N | NN

F + S | F + NS | NF + S | NF + NS

F + S + N | F + S + NN | F + NS + N . . . .

Class Explosion → n properties → 2^n classes

↳ Solution: $\boxed{\text{Interfaces}}$