SOLID - I

① SRP

② O | C

③ Real life case study

Abstraction
- └ P
- └ I
- └ E

$\left.\begin{matrix}\text{Poly} \\ \text{Inh.} \\ \text{Encap.}\end{matrix}\right\}$

Design principles

→ Creating codebase

→ what should be in your class?

→ Structured

SOLID vs GRASP vs CUPID

↳ peer reviewed

↳ simple to implement

S - Single responsibility (SRP)

O - Open-closed (OCP)

L - Liskkov subsitution (LSP)

I - Interface segregation

D - Dependency inversion

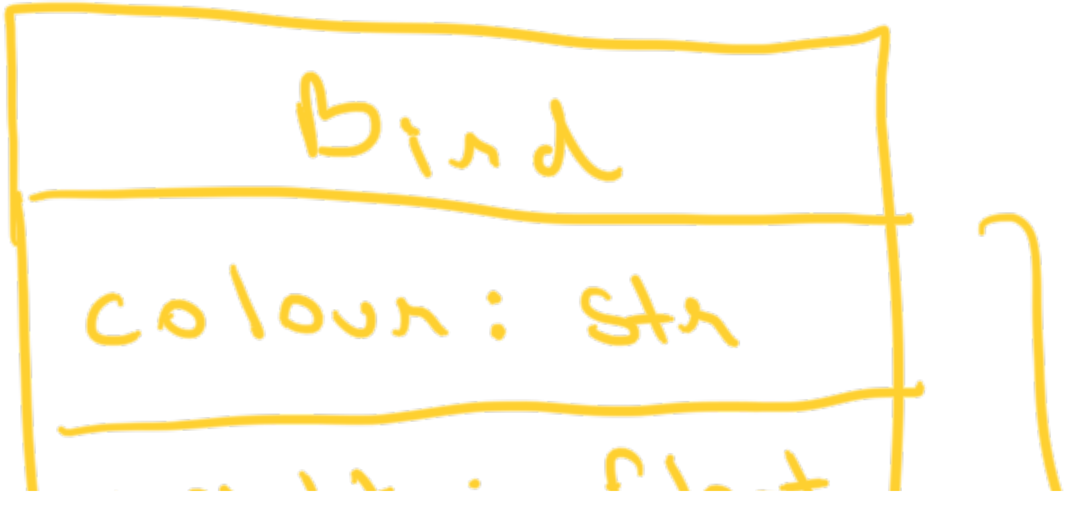
Case Study

Design a bird } Amazon

Not implement

v0 - Bird Class

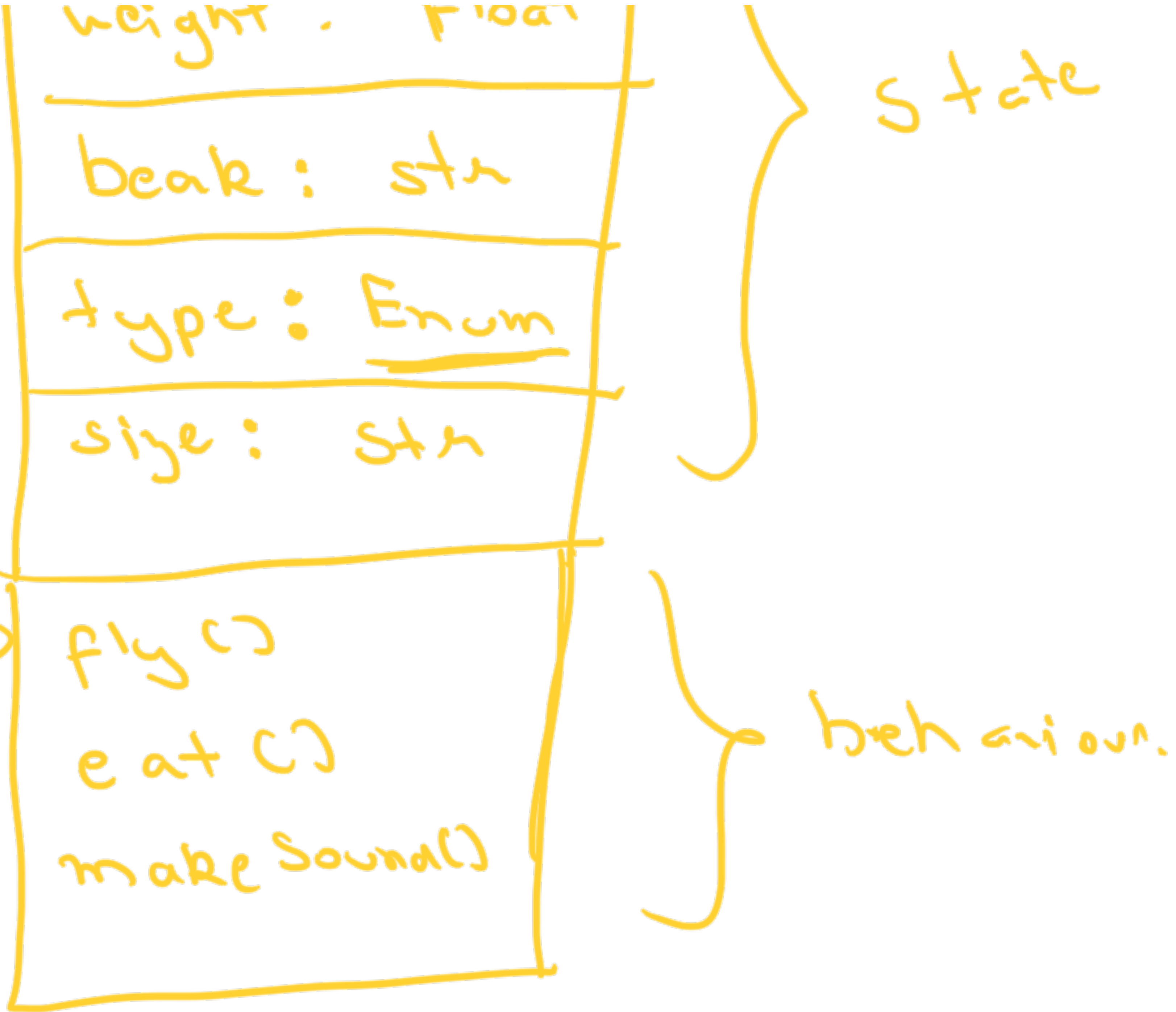
① Adding



common
behaviour/fields

② Add new
type of bird

③ update existing
behaviour of
any bird



$\boxed{\text{Parrot}}$

```
fly() {
→    if (type == "Eagle") {
```

```
          ┌──────┐
          │ .... │ ── fly like an Eagle
          └──────┘
  →  } else if (type == "Sparrow")
          ┌──────┐
          │ .... │
          └──────┘
     } else if ( type == "Parrot"
     ....
}
```

A → Eagle

B → Sparrow

Problems

Not

(1) Readable *

(2) Testing

(3) Maintainable - merge conflicts

(4) Reusable - make reusable

(5) Too many things happening in the same method - SRP

Single Responsibility (SRP

One code unit should have only one well defined responsibility.

$$\begin{tikzcd} \text{code unit} \arrow[d] \\ \text{class} \end{tikzcd}$$

method
package

Change - why are we making this class?

SRP - There is no right answer.

Places where SRP is generally violated

① (if - else) in the same method

→ buisness logic

Tax calculation

```
{        if ( income >0 && < 10L)
   {            .....
         else if ( income > 10 )
   {           .....
```

→ access control

```
{   if (role == "ADMIN")
          ...
    elsif ( role == view)
```

② Monster methods }
God classes

get

SaveToDatabase() {

{ ① [Query q = "Select * from monster";

② [User u = new User();

100 ③ [Db connection

execute the qu

close the connection

}

③ Helper / utils

junk yards for common code

Java - SRP

Python - LSP

java.util.

Collection

String

java. collection
. String. StringBuilder

SRP

1. Only one reason to change code

2. Where SRP
   - → multiple if-else
   - → monster methods
   - → utility methods.

3. SRP is subjective

5:57 : 6:05

10:35

basics

↓ → final

final

Design document

→ Publish → review

↓ ↓ ↓

Today → Tommorow → Someday

printInvoice ↓ ✗

inKotlin di

Sep. class

Over engineer

→ iteratively

+1 x2 } Reduce effort

+2 x2 } technical

Slack

→ # random

[illegible]

↑ ↑

→ utils .string

↳ Tokenizer

$\boxed{\text{SonarQube}}$

O | C | OCP

— open - closed principle

My class should be open for extension

but closed for modification.

Add new functionality

↳ not modify existing code

very less modification

ideally, no code

Benefits

① Testing

② Less chances of bugs
Regression

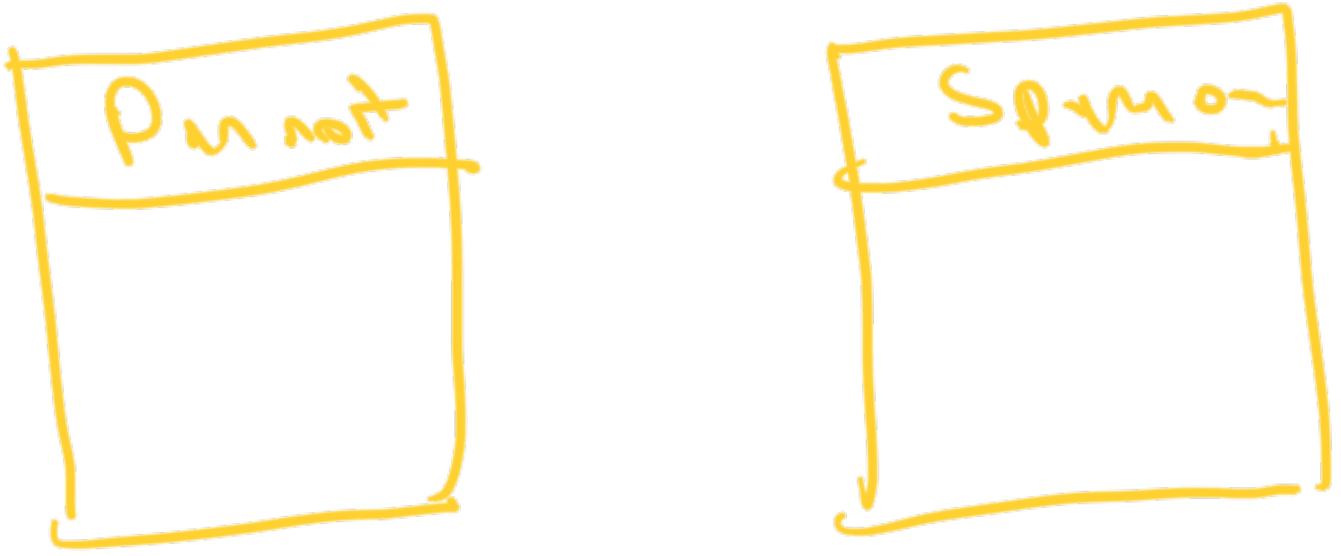
How can we fix our Bird class?

Keep common in one class
+
Specific in another class

Inheritance

interfaces?

Bird
weight
type



Interfaces & Abstract classes

→ blueprint for behaviour.

interface Animal {

No impl

```
public void makeSound();

}

class Person implements Animal {

      makeSound() {
         Talk all day
      }
}
```

Abstract classes

↑

[illegible]

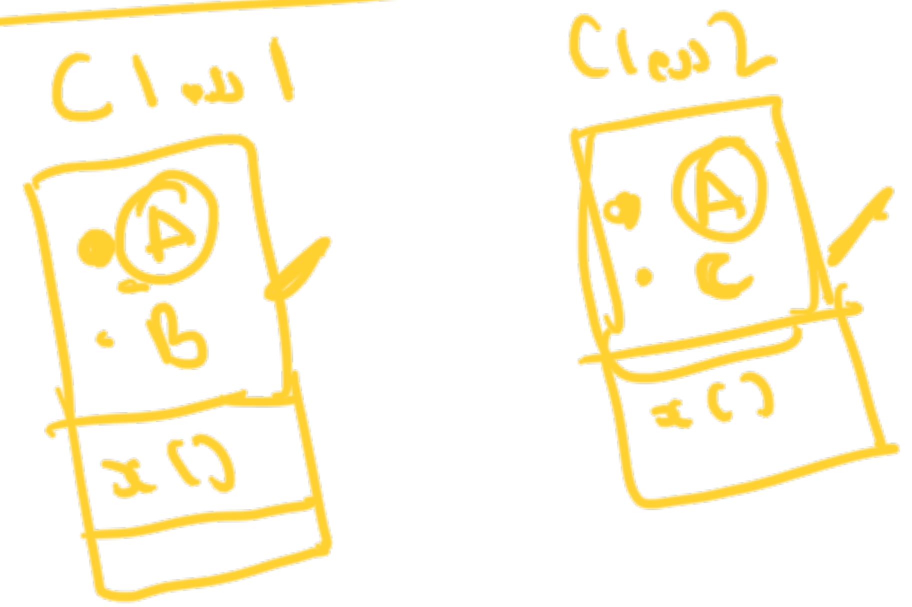
hybrid → AC state + behaviour

→ abstract methods

↓

not implemented

AC → no instance



Redundant code / attributes

Inheritance

Inheritance

① Do 2 classes have common functionality?

Yes

② Do they have common state?

→ Abstract class

Abstract Classes

→ Create an instance of the parent class

→ when behaviour needs not be implemented

Bird - v2

→ one class

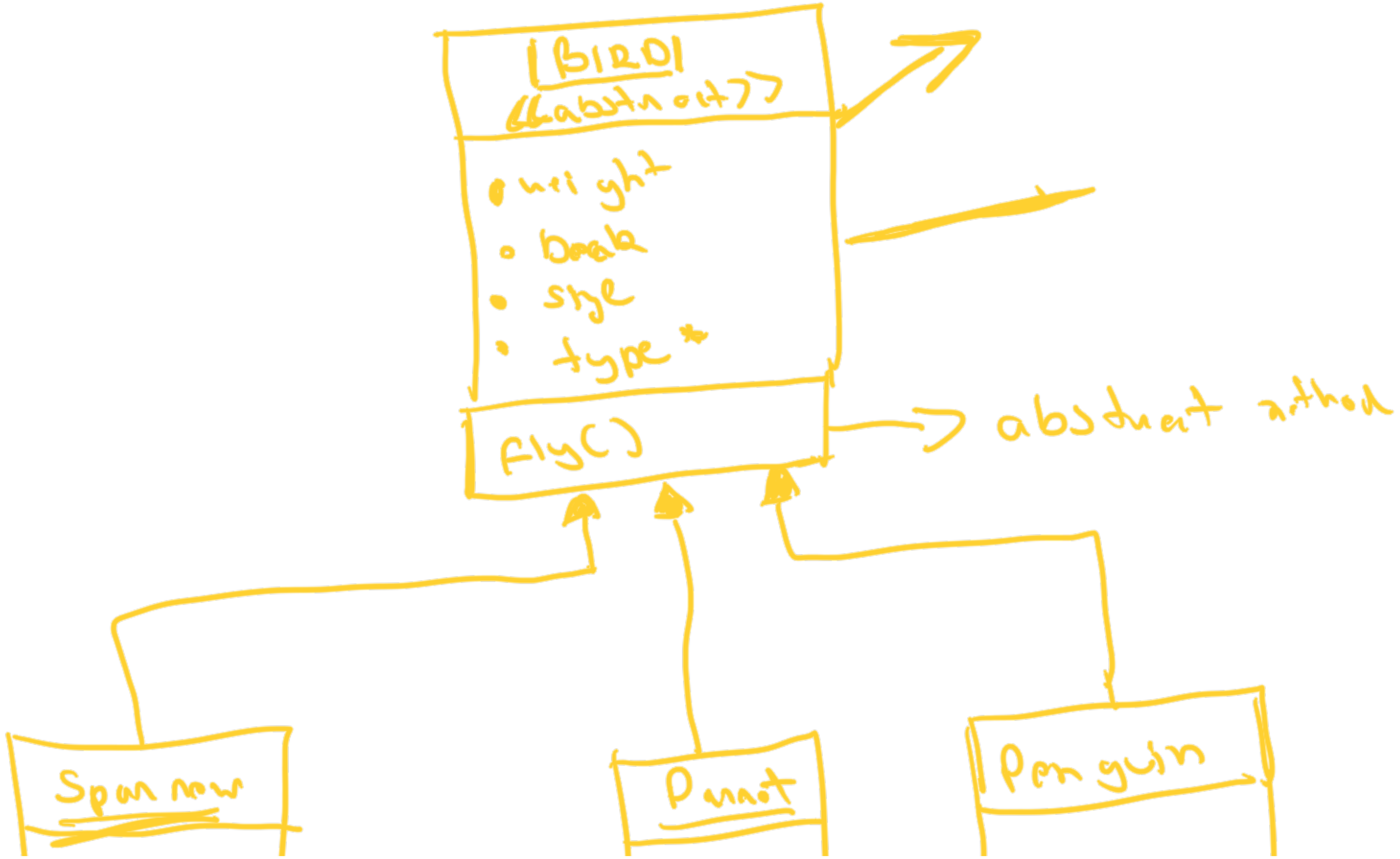
→ violating SRP + O/C

Sparrow vs Parrot

→ common function ✓

→ common state → Te8

→ Abstract class



1. throw an exception
2. return;
3. can add isfly

```
setBirdsFree(List<Bird>) {
```

```
for each bird:
        bird.fly()
}
```

Penguin

→ special handling

Liskov

→ special handling for child class

A class should only have methods it needs to be implement

$\bigcirc\!\!\!\!\!\!\text{v3} \longrightarrow$