SOLID Design Principles

Agenda

- Interfaces (10-15 Mins)
- SOLID
  - SRP (Single Resp Principle)
  - Open/Close Principle
  - Liskov Substitution Principle
  - Interface Segregation Principle

→ Dependency Inversion

↳ Diff B/w

Dependency inversion	Dependency injection
inversion of Control	

INTERFACES

Class : Blueprint of entities

Interface : Blueprint of a behaviour

interface Runnable {

    void run();

}

interface List {

    void add(Object);

    void remove(int);

}

int size();

}

class ArrayList implements List {

int age = 10

int count = 23

void add (Object o) {

}

void remove (int index)

}

Interface

1. Blueprint of a behaviour
2. By defining an interface we basically say that anyone who has the methods listed in the interface def^n

is valid for us.

(3.1) interface Name {

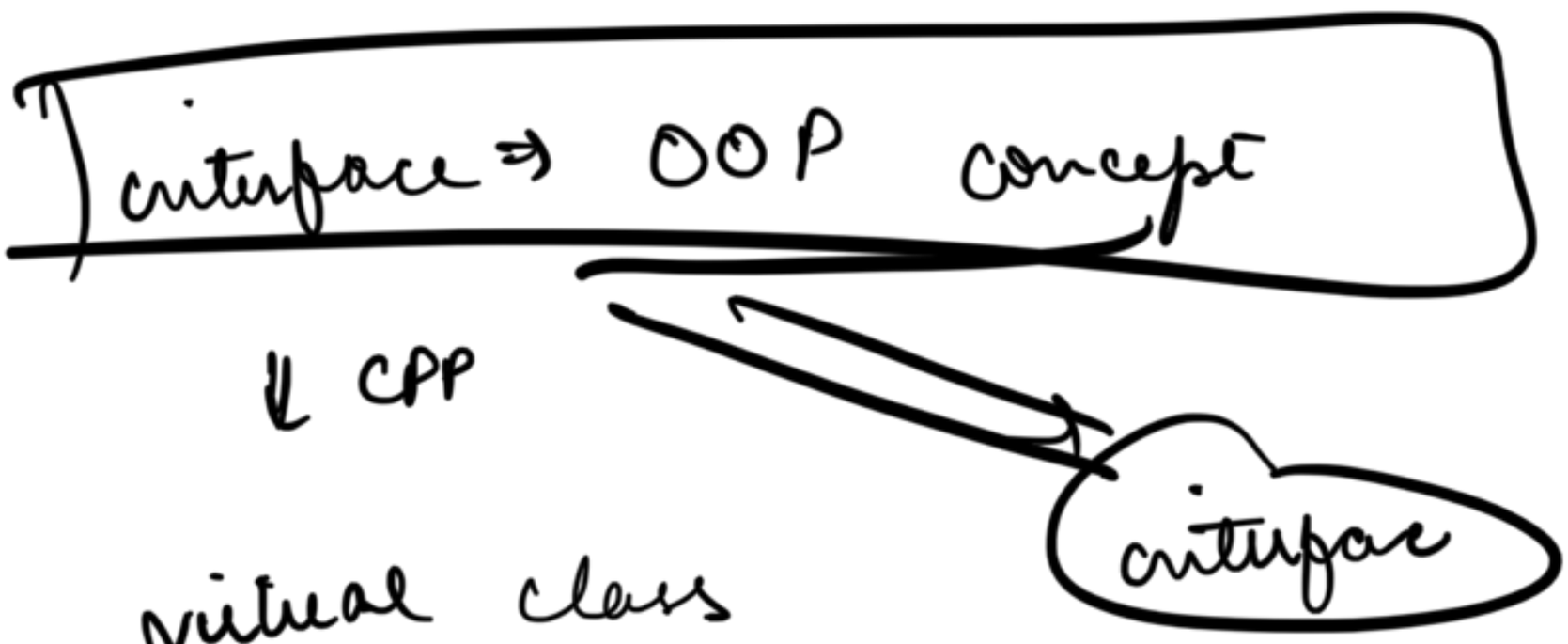
void add();

int remove();

}

class Arraylist implements Name {

}



Python

→ RBI banned Yes Bank for

Online transactions

UPI Payments

Phone Pe used Yes Bank as its service provides

< 1 day for them

class PhonePe {

ICICI Bank API

String

YesBank API

YB API( )

1

unit

YBapi.getCardDetails( )

makePayment( ) {

refund( ) {

}

getBalance(); }

updateKYC();

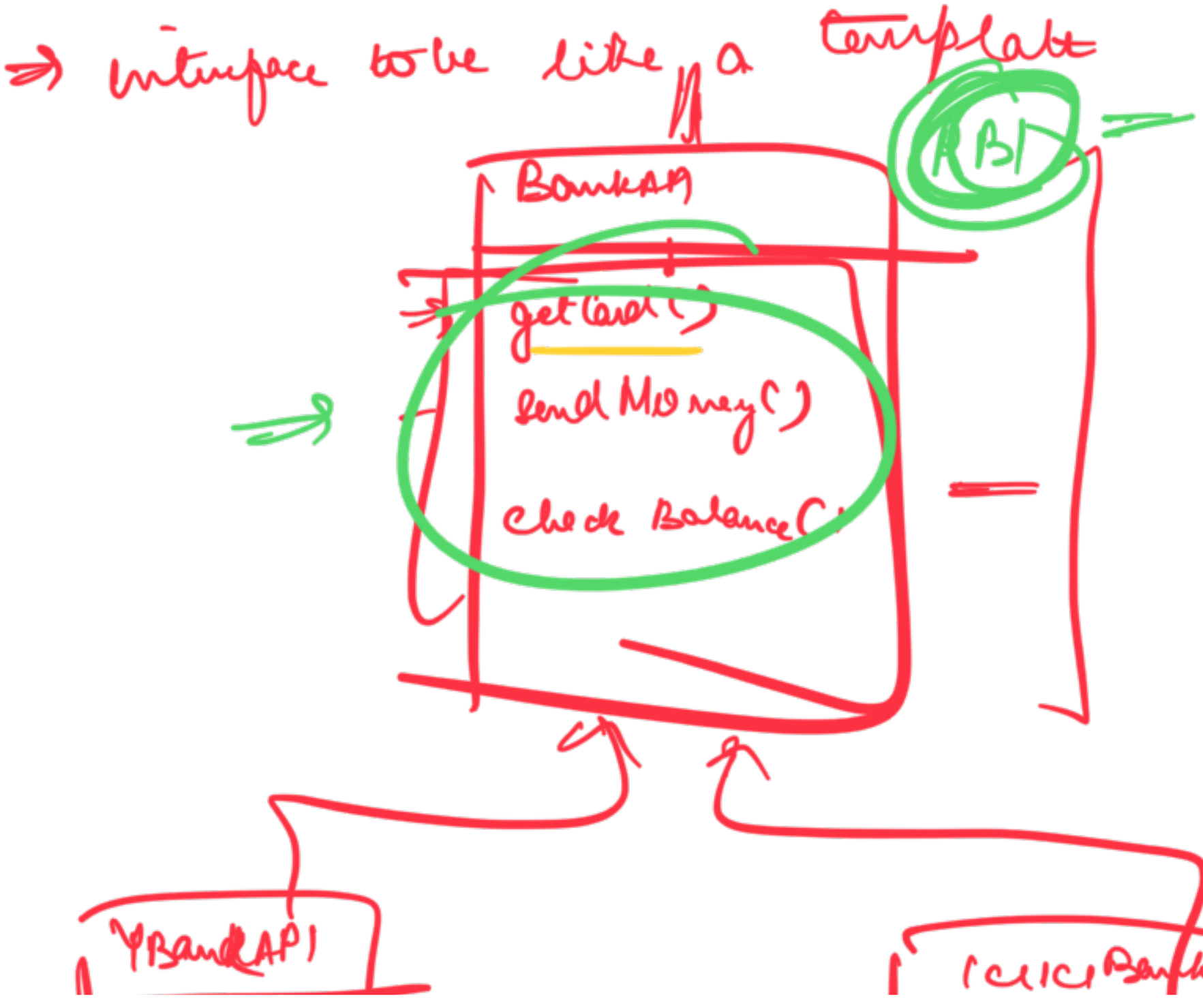
transferMoney();

}

Both of my service providers might have totally different APIs (the set of methods and their return types)

⇓

going to be very difficult to interchange


⇒ interface to be like a template
RBI
BankAPI
getCard()
sendMoney()
checkBalance()
YBankAPI

getLevel()

SendMoney()

CheckBalance()

getLevel()

SendMoney()

CheckBalance()

Polymorph

PhonePe

YBAPI YBAPI()

PhonePe

BankAPI

new YBAPIC();

BankAPI

{

}

YBAPI implements BankAPI {

}

BankAPI API = new ~~YBAPI()~~

ICICIBankAPI

api.getCard( )

Dog

Animal a = new Dog()

Parent Class

Animal

child Class

Dog

interface

BankAPI

implementation

YB API( )

class YB API implements BankAPI {

Animal a = new Dog()

a.

BankAPI b = new YBAPI().

}

BankAPI b;

if (config.bank == "YES")

b = new YBAPI()

Adapter Design Pattern

```
if (config.bank == ICICI)
    b = new ICICIBank()
```


→ can extend ONLY one class

but I can implement no of interfaces.

interface & abstract class

OOP interface is like a concept

Java interface => how to imp OOP interface in Java

=> Diamond Problem

List

→ diversity

Values of good

Software design

↳ Not how to implement those

values ⇒ Design Patterns

Set of 5 principles that should be followed in any good software design -

Design a Bird ← Amazon interview Problem

```
class Bird {
    String color
    int weight
    int flyHeight
    int age
    fly() { }
}
```

eat() {

walk() {

}

fly() {

if type == crow

else if type == hen

else if type == penguin

}

else if type == sparrow

Unit Testing

1. Difficult Testing
2. Difficult to understand
3. Difficult to work on parallel

It is in violation of SOLID

Single Responsibility Principle

Any code unit should have EXACTLY ONE

RESPONSIBILITY ⇒ Exactly one reason to change

Method / class / project

Resp :

1. To maintain attributes and behaviours of every type of bird
2. How every type of bird will fly

How to identify SRP violation

1. if method has multiple if / else.

(2) Monster Method

→ That is doing more than what its name is asking it do

leaving the code to do

print ( ) {

SQLQuery = " "

'Query += " Select "

Query += " c from xyz where "

db - get Result (query)

createQuery

print (result)

}

createQuery() {
SQLQuery = "
SQLQuery +=

}

getResult() {
Qy = createQuery()
return db.getResult(q)
}

print() {

→ MD2 number.

print (getResult())

)

3 Util Packages

⇒ Can become a place for every random thing

util - Date Utils
util . Time Utils
util . Cal Utils

⇒ time . Date
time . Time
time . Cal

⇒ It is discouraged to have classes in packages with util in

getDate()
getTime()
getCode()

util . Utils

name -

Date

data transformer

java.util.Date

Collection

Set

Current

java. collection. List

. Map

. Set

java. util

java. currency.

java. time.

following 100% SRP is not possible.

① Every code unit → EXACTLY ONE REASON

only one reason

② Identify SRP violations

① if-else

② Monster Method ⇒ Readability

③

③ Utils.

Why follow SRP

① Reusability

② Readability

③ Testing

④ Multiple people to work in parallel.

⑤ almost always solves O of SOLID

⇓

Open Close Principle

Break till 10:45

O ⇒ Open Close Principle

Codebase should be open for extension

but closed for modification.

while adding a new feature, My CS should be open for extension (easy to add new feature), but closed for modification (I shouldn't be needed to make much changes in already existing codebase)

HDFCBANK

@abstract Bird

color

age

weight

type

flyHeight

abstract fly()

eat()

abstract walk()

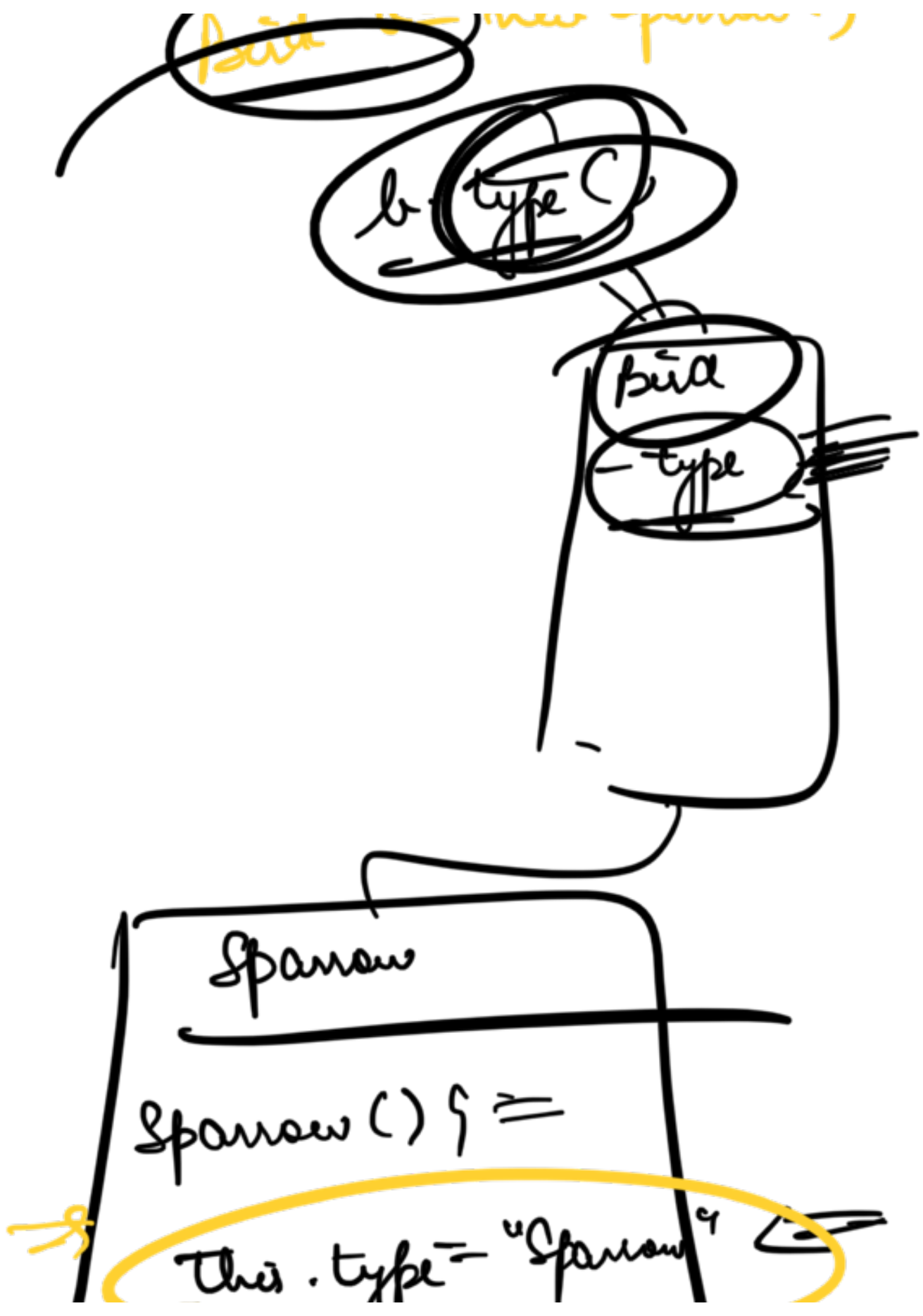
Sparrow

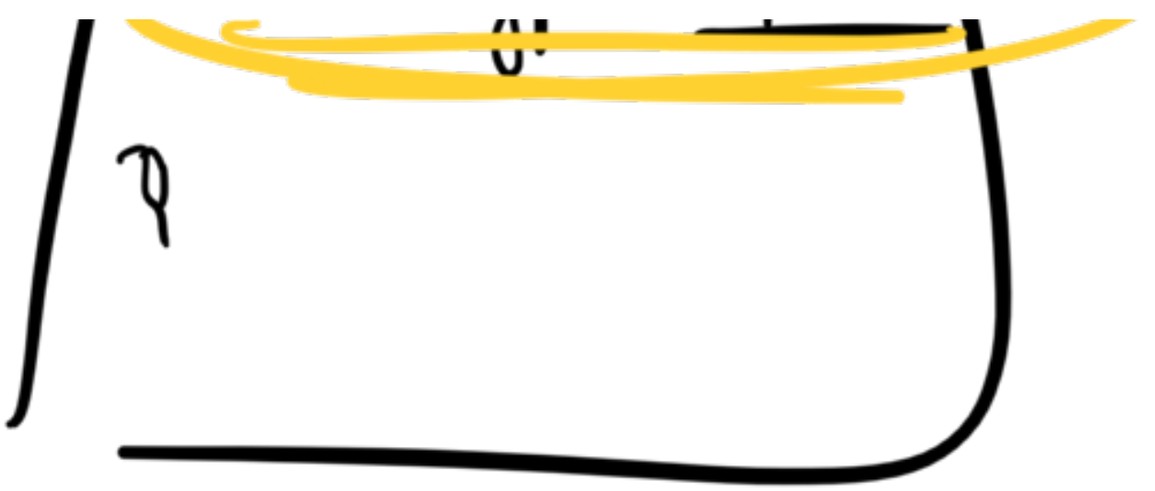
Crow

Region

Hen

Fungus


bird
bird type
bird
type
Sparrow
Sparrow() {
this.type = "Sparrow"



fly method of Penguin

① Provide a default impl in Bird

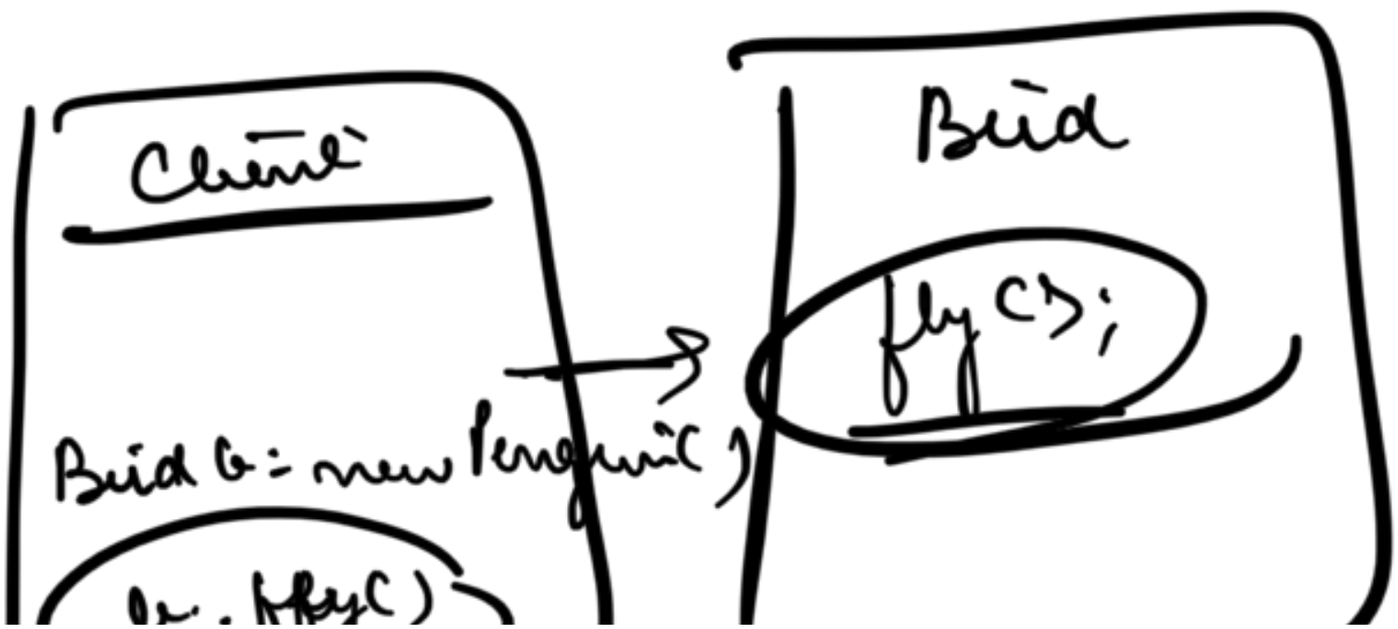
② Throw an exception from child

③ Keep the Method blank

④ Return a special value (NULL)

⇒ If we provide a default method in bird,
if we forget to implement the method
in a normal class ⇒ can we

Throw an exception for child



Good coding

Everyone can fly

Client {

List<Bird> birds = [Penguin, Crow, Pigeon]

for (Bird b : birds) {

if (b.type == "can fly")

Errors should be encountered at

Compile time not

Runtime

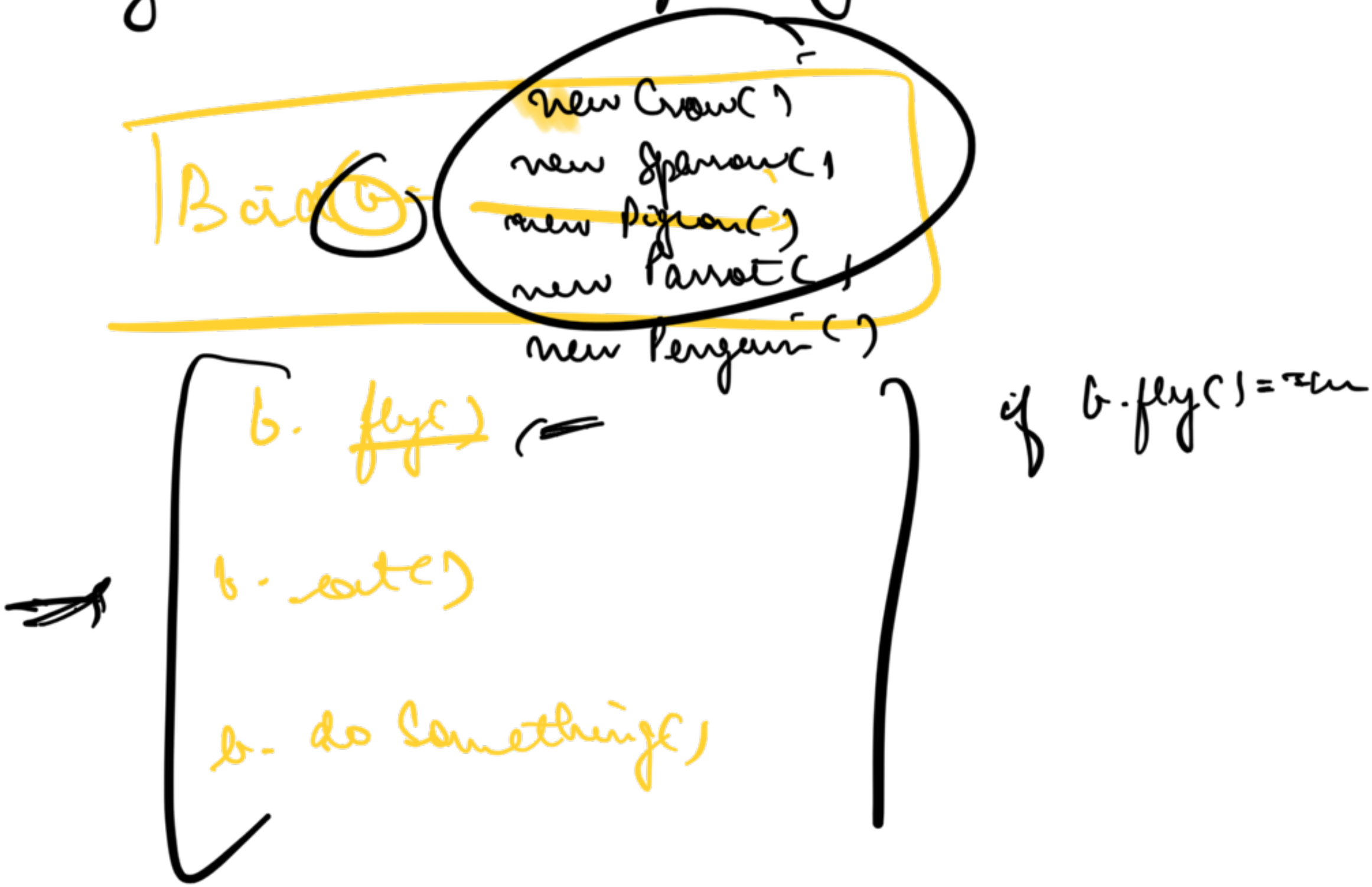
(L) Liskov's Substitution Principle

Any variable of parent class should be

As-is able to be substituted by

an object of any child class without

my codebase needing any modifications.



No child class should get special treatment

⇒ If your CB is requiring special treatment
↳ BAD DESIGN

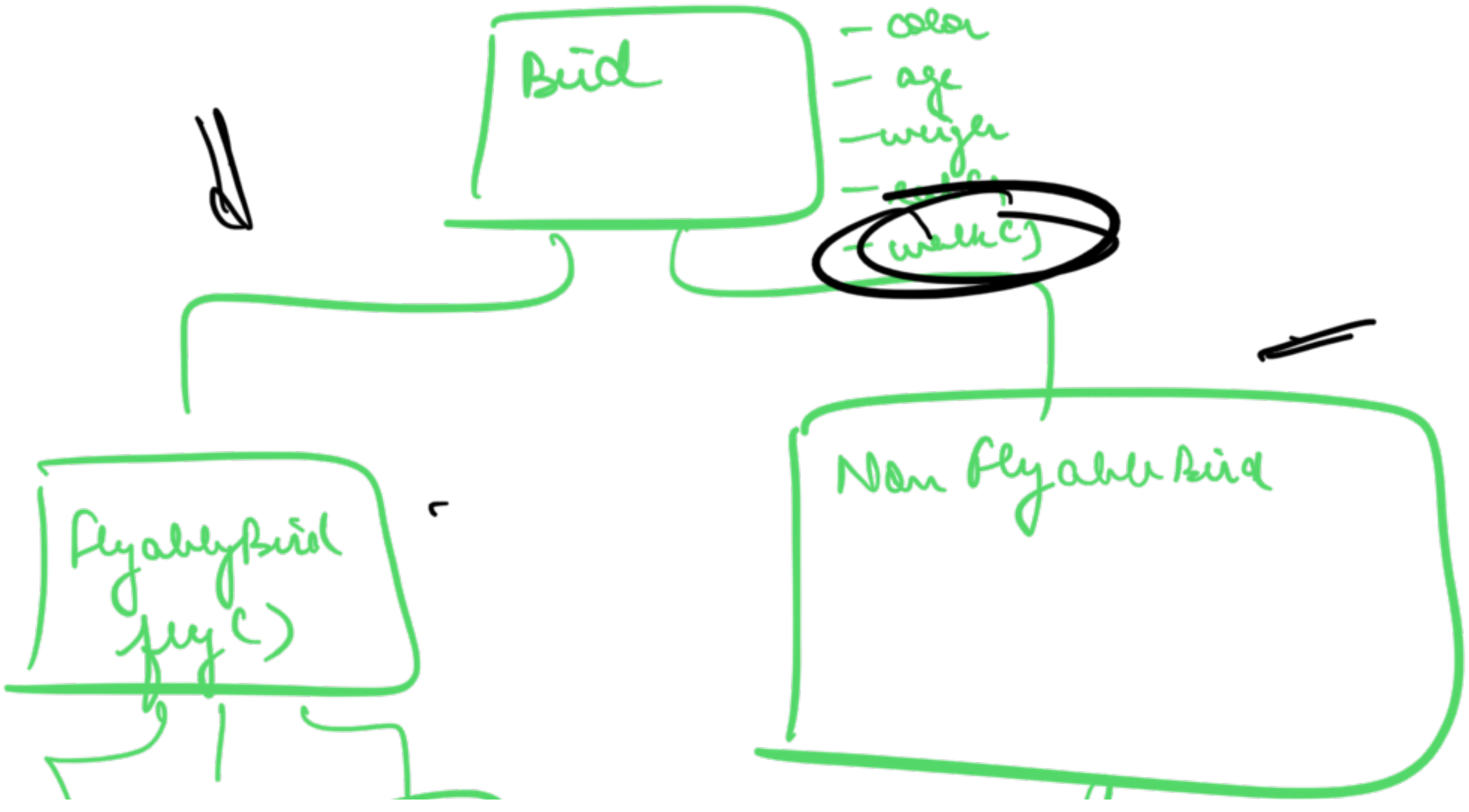
we have 2 Types of Birds

① Who can fly → Crow
→ Sparrow

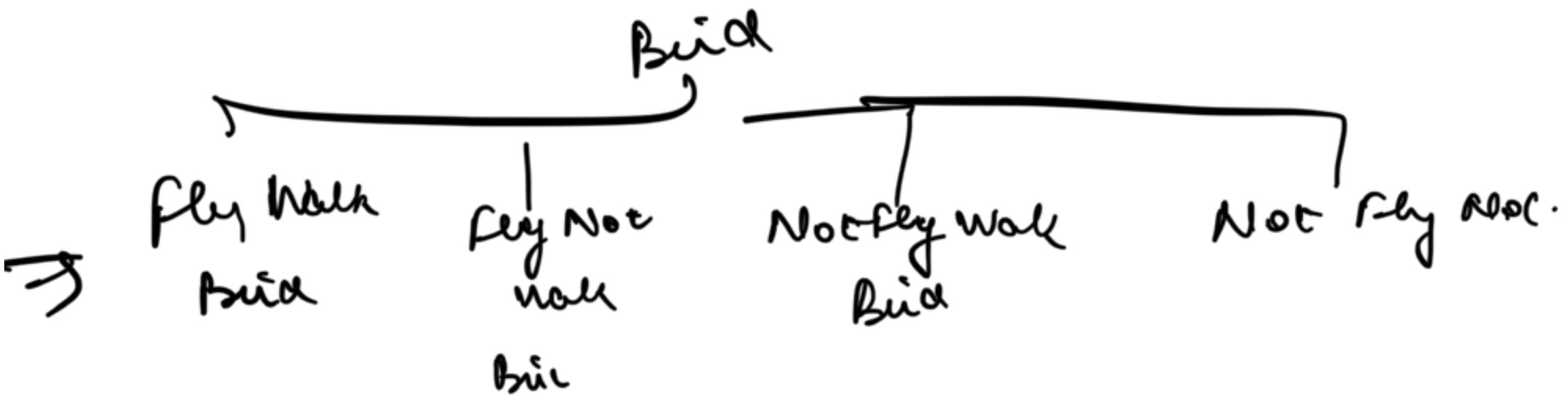
② Who can't fly → Pigeon

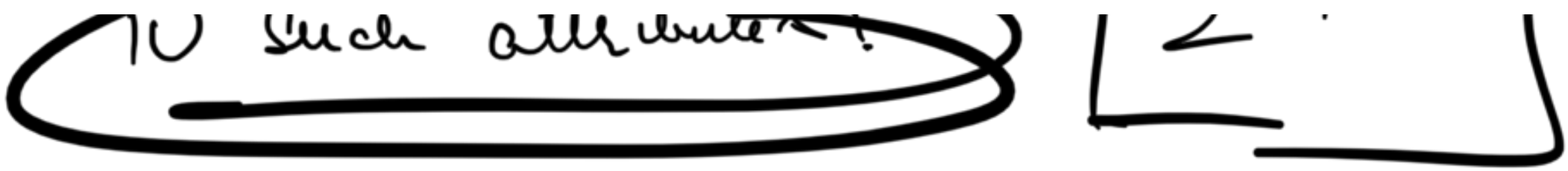
↳ Penguin

AIM: I want to check issues at compile time



List < FlyableBird >




Such attribute?


Fly
Not fly