LLD Contd: SOLID Design Principles

Rules / Principles for a good Software

Human Principle
→ Ethical
→ Hard Worker

Design Principles: values / qualities / rules that should

be ideally followed for a
Software design

SOLID Design Principles

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

SINGLE RESPONSIBILITY

class Bird {

type:
blob type
height
gender

void fly () {

if type == sparrow

else if type == crow

else if type == pigeon

}

}

- ① Details of features of a bird

② Logic on how each type of bird should fly

Single Resp Principle

Class / Method / package / project / ser.

⇒ Every code unit should have a single well defined ownership

How to identify violation of SRP

① Multiple if-else / Switch case

(condition

② Monster Method | Class

→ does a lot more than what its Name says

Monster Method

fly() {

→ Connect to DB

→ Some logging

→ write a query

→

③ Utility Packages / Classes

⇐ Common / help

⇐ Become a junkyard of multiple types of things

violates SRP

Util ⇒ package

↳ connects to Database ⇒ com.scalr.project.database.connector

→ DateUtil → com.scalu.project.dateUtil

→ Parser (XML | JSON) com.scalu.project.parser

→ Token

→ Third Party Service

⇒ JAR

used to feel like.

Save User And Log CSV

}

Why is SRP imp

What will go wrong if SRP not follow :

1. Code Reusability
2. Packages / classes will become bloated ⇒ Understand
3. Testing

w/ SRP have more classes / package / f

Following any DP 100% is not practical - You have to decide on a trade off

Clean Code

≈ 10 min

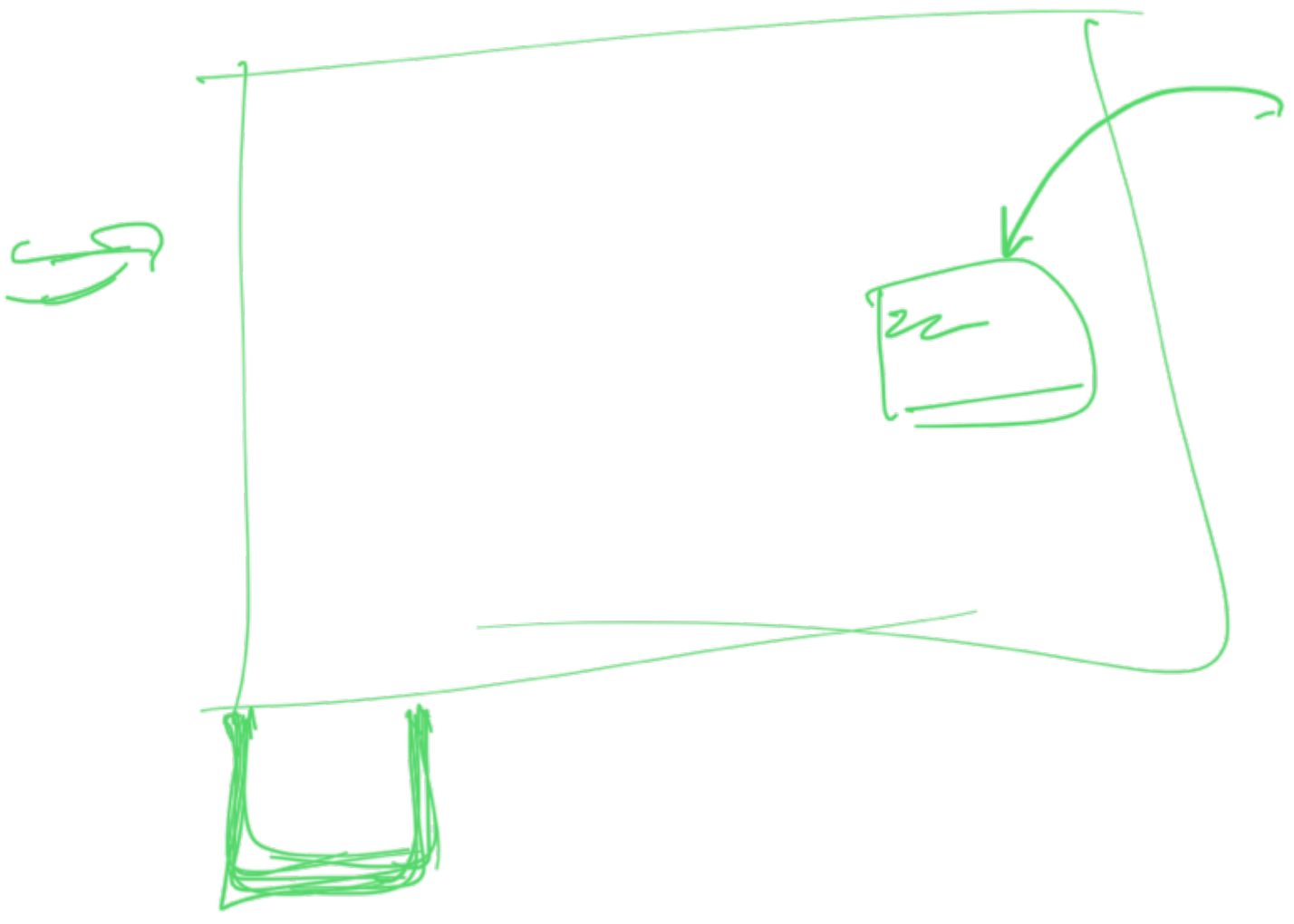
O: Open close Principle

⇒ Any code unit

should be open for extension

But closed for modification

When adding new feature



void fly () {

elif type == pig

elif type == sparrow:

elif type == crow:

elif type == parrot:

}

Modify existing CB

Modify Tello

Present Design

Bird {

type

color

weight

gender

fly()

color

abstract fly();

Pigeon

Sparrow

Crow

Parrot

Move flings
take 2 steps ahead
wait next

Code Duplication if behaviours Shared

interfaces

Inheritance

Bird
type
color
gend
fly();

interface
Flying
Behaviour
fly();

Sparrow
fly() {
WhiteFlyingBehav
fly()

Crow
GreenFlyingBehav
fly

Pigeon
WhiteFly

WhiteFlyingBehaviour
fly() {

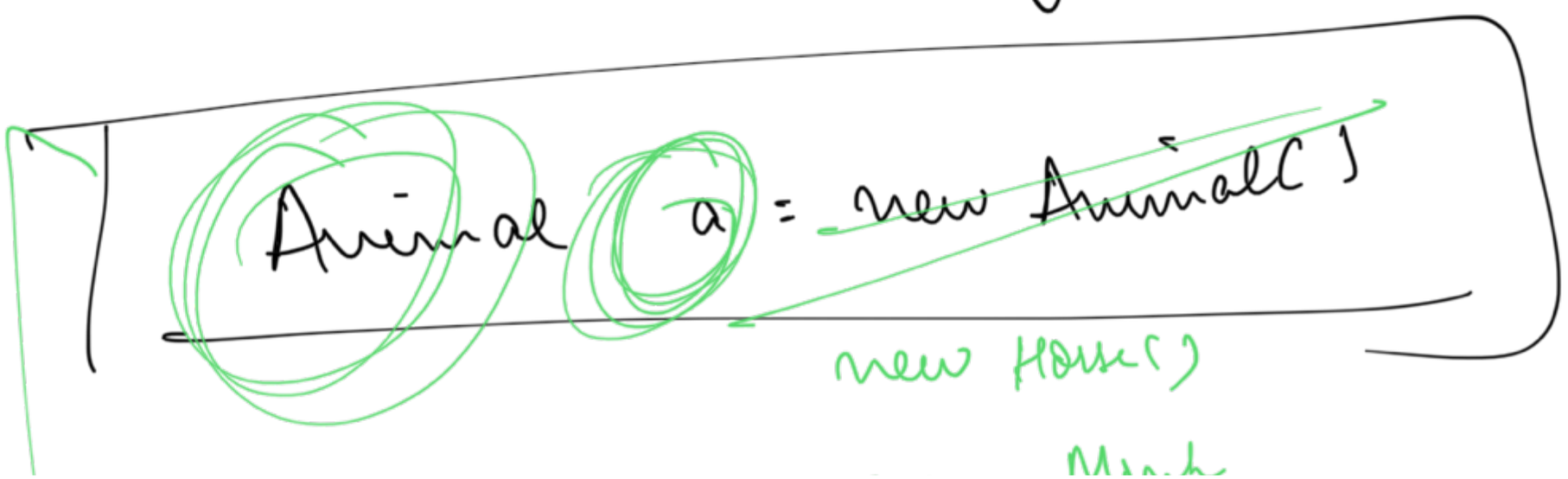
GreenFlyingBehaviour
fly() {

Bird
gender
type
color
flyingBehaviour
flyCS{
flyingBehaviour.fly
}
Composition

favour composition instead of inheri

(2) Liskov's Substitution Principle

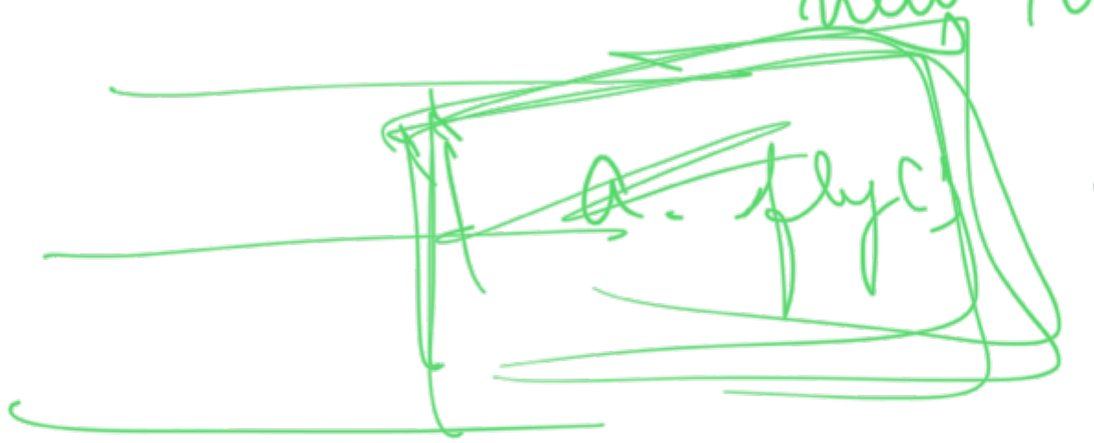
(2) ⇒ Any parent class should be as is replaceable by an instance of a child class without any special treatment needed



new

new Butter

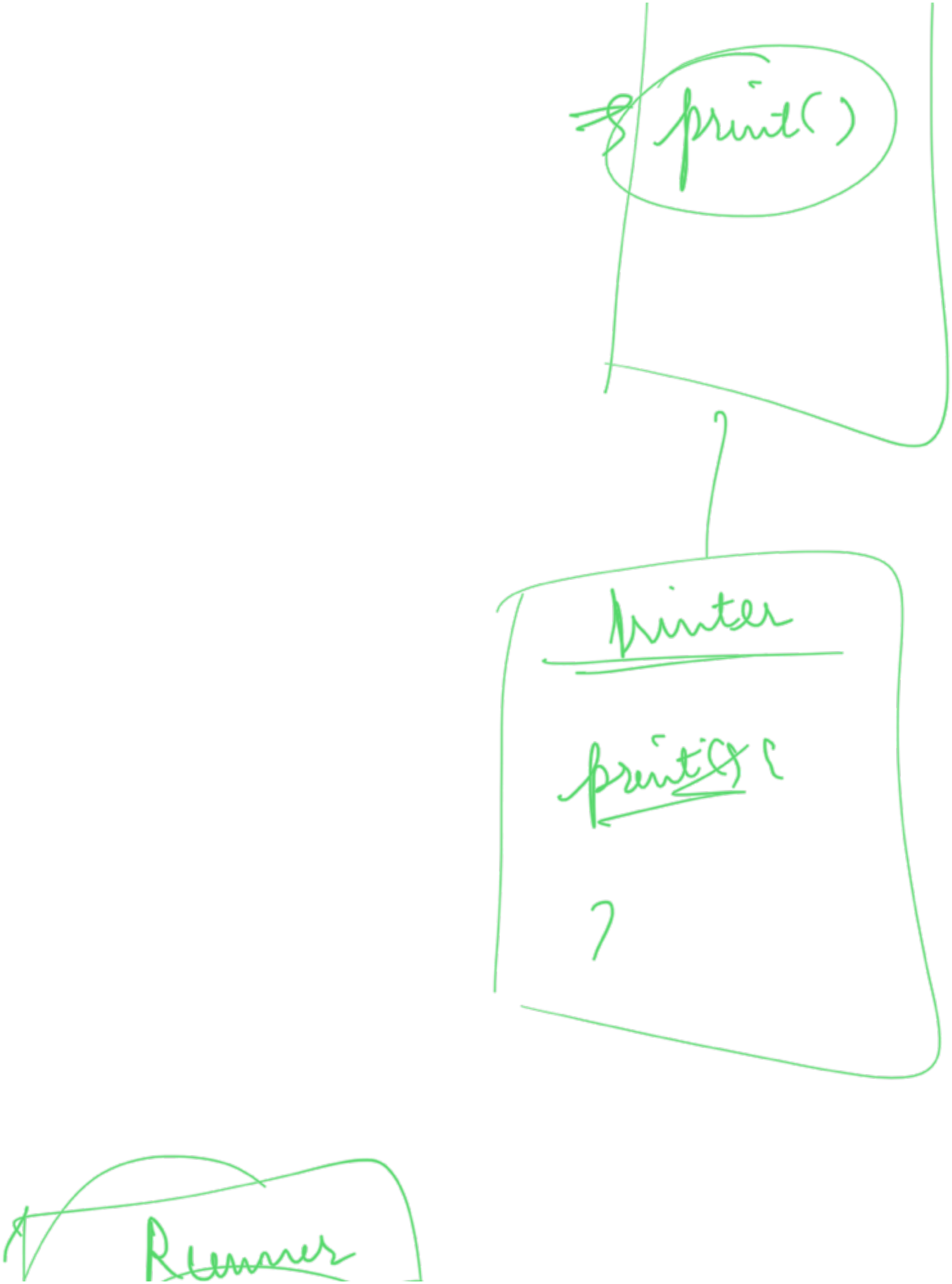
new Human



Child class should not give a diff output/
add diff behaviour to a method of the
parent

Class

Console Printer


print()
Printer
print() {
}
Runner

run():

Thread

run() {

}

(contd...)

Child class should not give a new meaning

→ to a method of the parent class

→ Add any other behaviour (eg throw exception

→ Do anything else

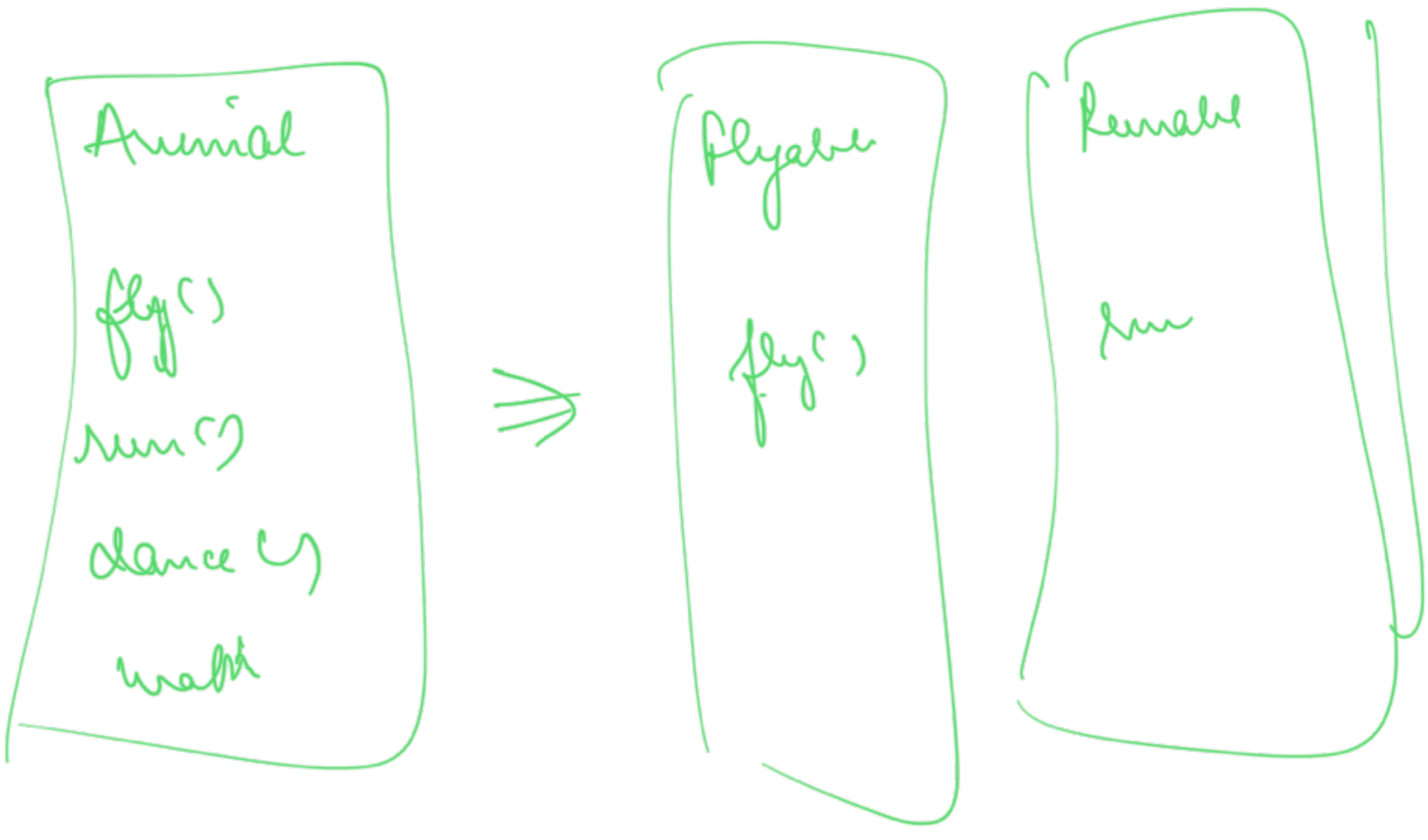
⇒ Avoid surprise for the client

I ⇒ Interface Segregation Principle

⇒ Don't have thick/ generalize/ fat interfaces

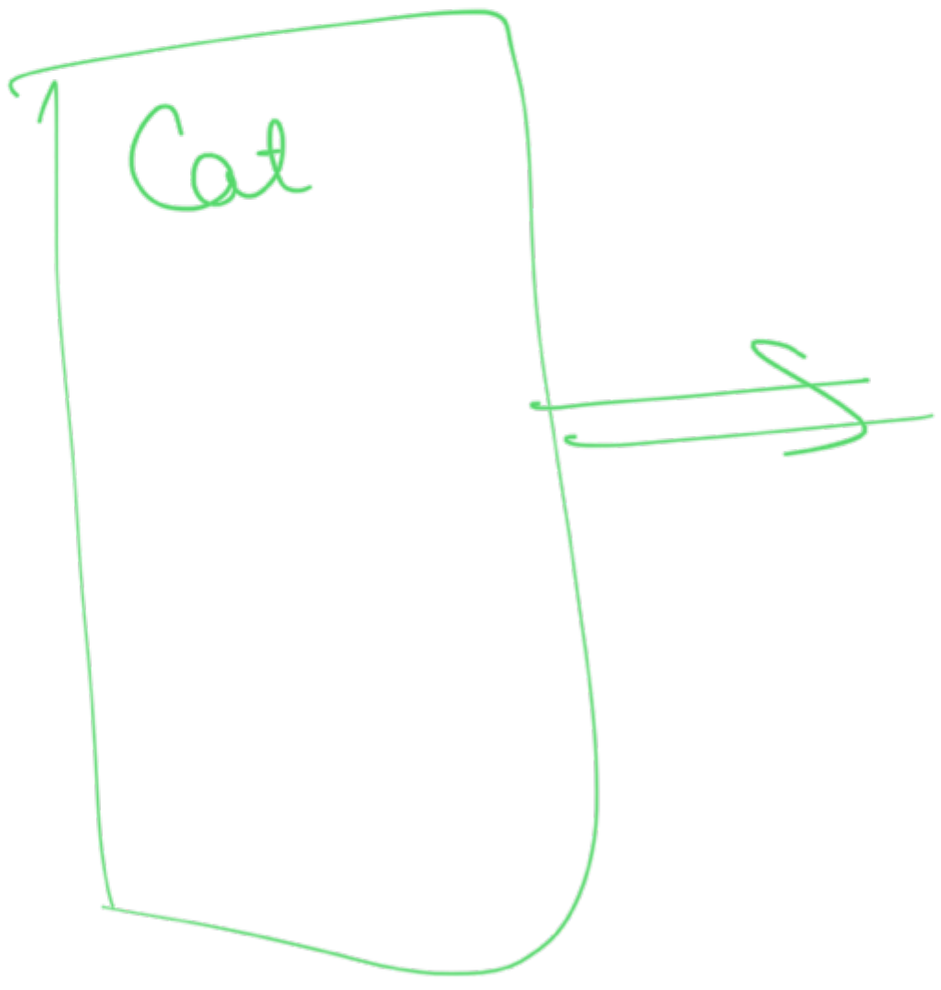
→ Make interfaces specific

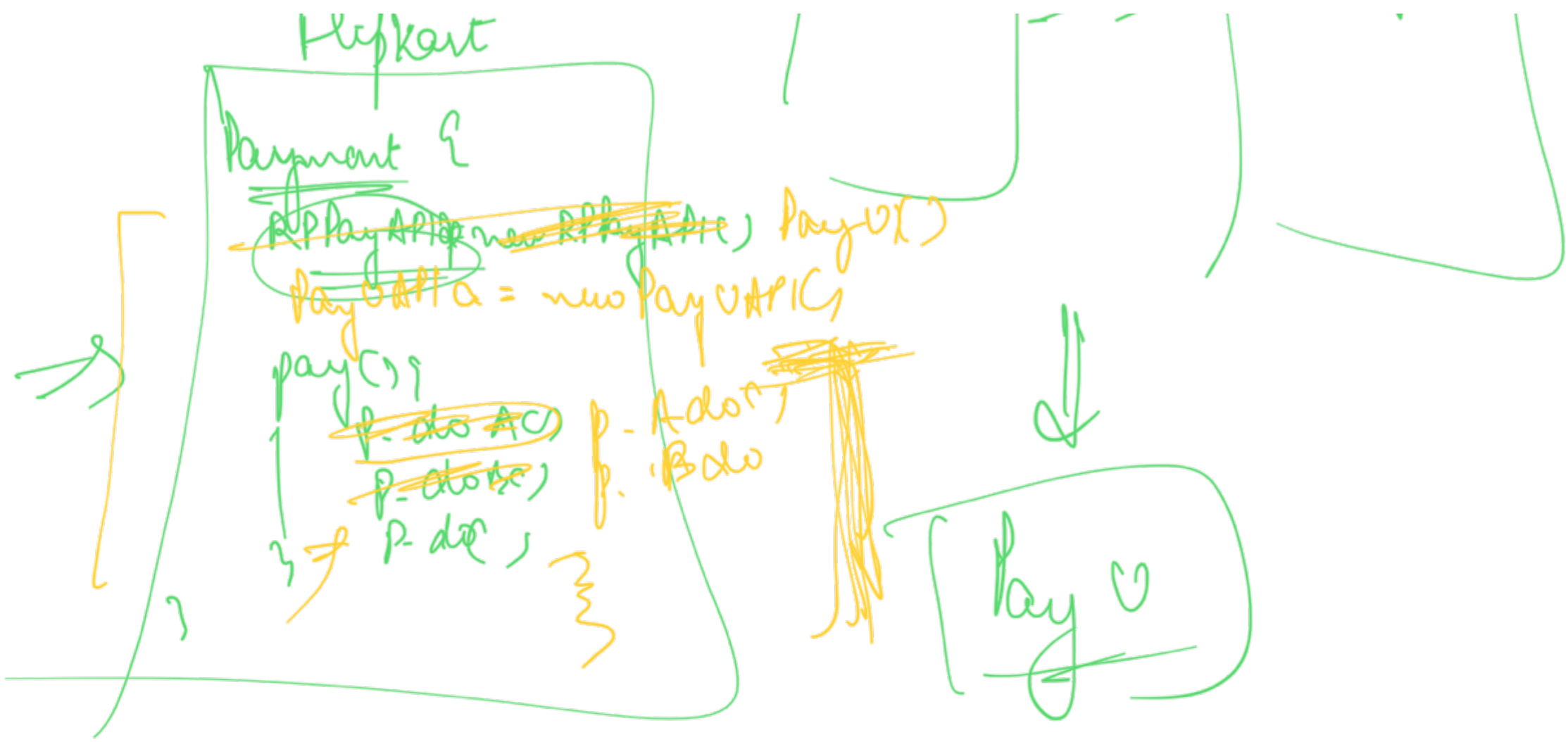
→ Have interfaces with as less methods as possible



D ⇒ Dependency Inversion

⇒ No two classes should be directly dependent on each other. They should be connected via an interface.





A testing hazard

Payment
Payment API Interface

No 2 Concrete Classes should depend on

each other.

Code to an interface not to an implementation
(Concrete file)

SOLID

D → Dependency Inversion

Dependency Injection

⇒ DBMS

⇒ LLD

Abstraction == Idea

Bigger the idea ↑ Bigger the abstraction

⇐ Highest level of abstraction

← Lower level of abs

Conclusion

Money → ① You inherited it from parents ← Inherit

② You created me ← Computer

Payment

RBPayment

Composition → HAS-A

Payment HAS A RBPayment

Inheritance → IS-A

Dog IS A Animal

PhonePe IS

⇒ YesBank → Banned By RBI from Online Payment

Phonepe → Yes Bank

Phonepe stopped working

→ 1 day

Yes Bank

ICICI Bank