LLD: SOLID Design Principles

OOP ✓

Design Principles

Ethics / Values

Software Design

Qualities that a great software design should have

→ SOLID Design Principles

S: Single Responsibility Principle

O: Open/Close Principle

L: Liskov's Substitution Principle

I: Interface Segregation Principle

D: Dependency Inversion Principle

(NOT INJECTION)

Case Study

AMAZON

Design a Bird

VO — Karthikeyan — Shuvrian

Bird

- weight
- colour
- type
- size
- beakType

- fly()
- eat()
- makeSound()

```
Bird sparrow = new Bird();
```

Behaviour of any bird changes

Common attributes/ of Birds — Methods

```
fly()
{
    if (type == sparrow) {
        ...
    }
    elif (type == crow) {
        ...
    }
    elif type == penguin ):
        ...
    }
```

PROBLEMS

① Bad for Readability

② Testing is Difficult

③ Difficult for multiple developers to work in parallel.

④ Violates ⑤ Single Responsibility Principle

⑤ Reusability is a problem

(Method / Class / Package)

→ Every code unit in your codebase should have EXACTLY ONE well defined responsibility

EXACTLY ONE reason to change

How to identify violation of SRP

1. A method with multiple if / else

Eg. when multiple if / else on business logic =

Given a number x, tell if it is a leapyear or not

2. Monster Method

Methods that are doing a lot more than what their name is suggesting them to do.

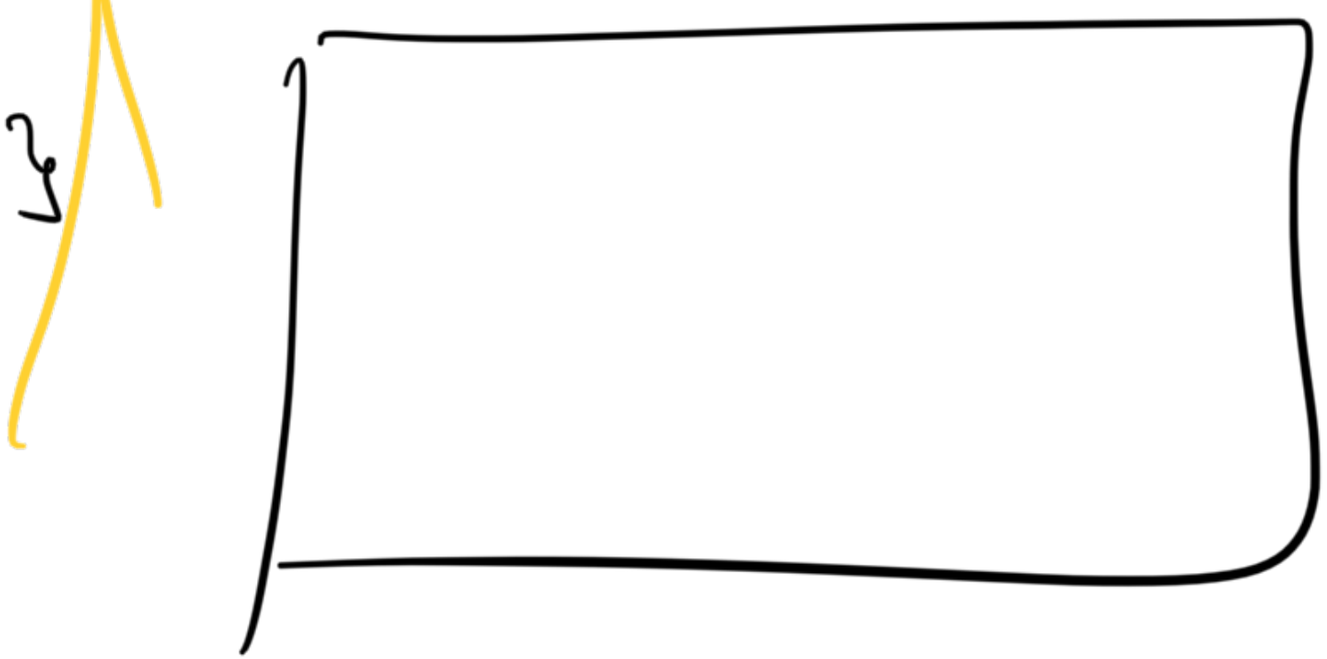
⇒ Save To Database (User) {

String query = " Select * from ---
--- "

User uObj = new Obj();

uObj.x = user.↑

Database Conn Conn :



Creat Query To Save Users {

}

Create Db conn() { }

⇒ Save To Database() {

}

(3) Utility Classes / Helper Classes

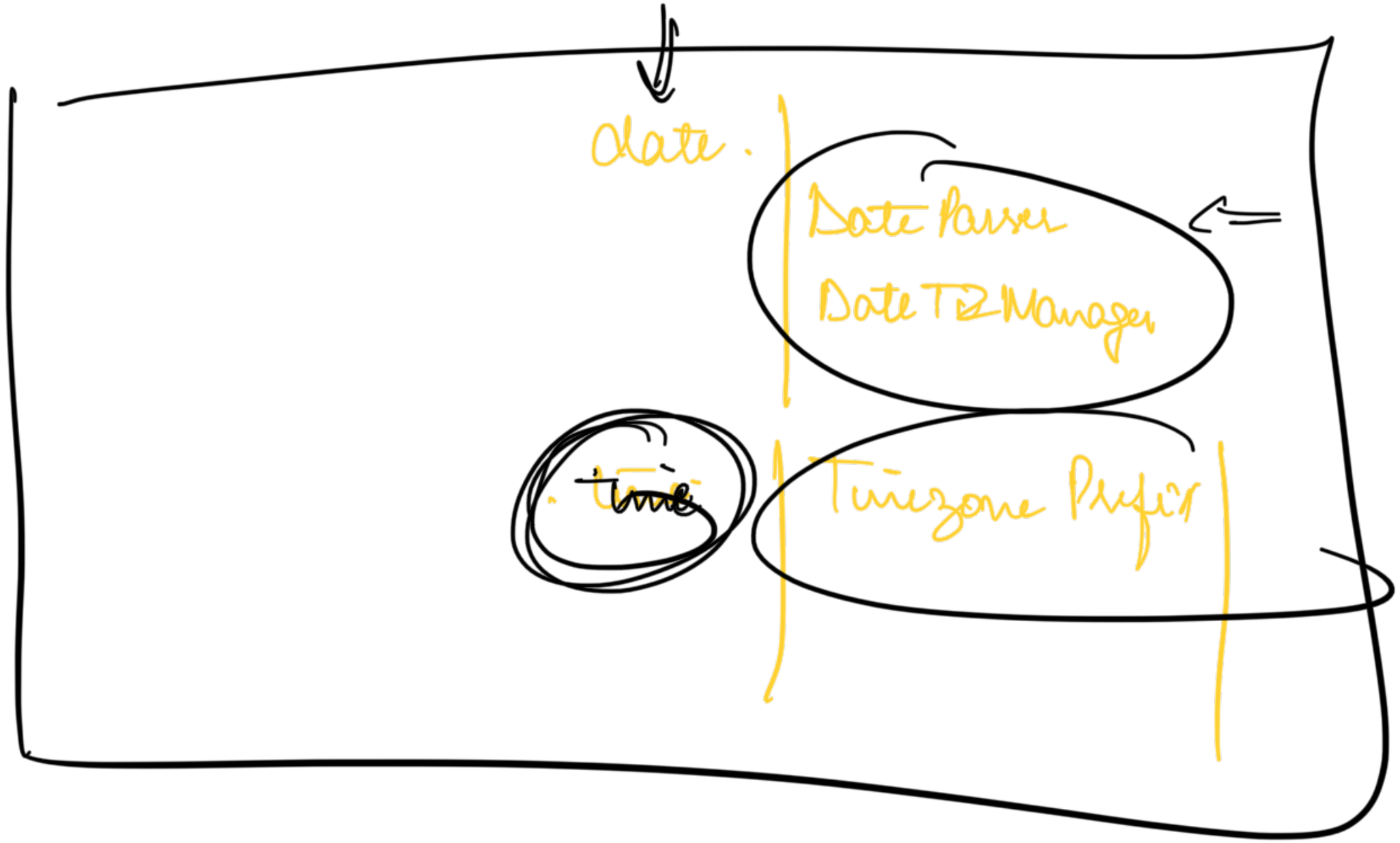
⇒ These packages / classes end up being a junkyard for a lot of unrelated code

util

Date Utils

Time Utils

Calendar Utils



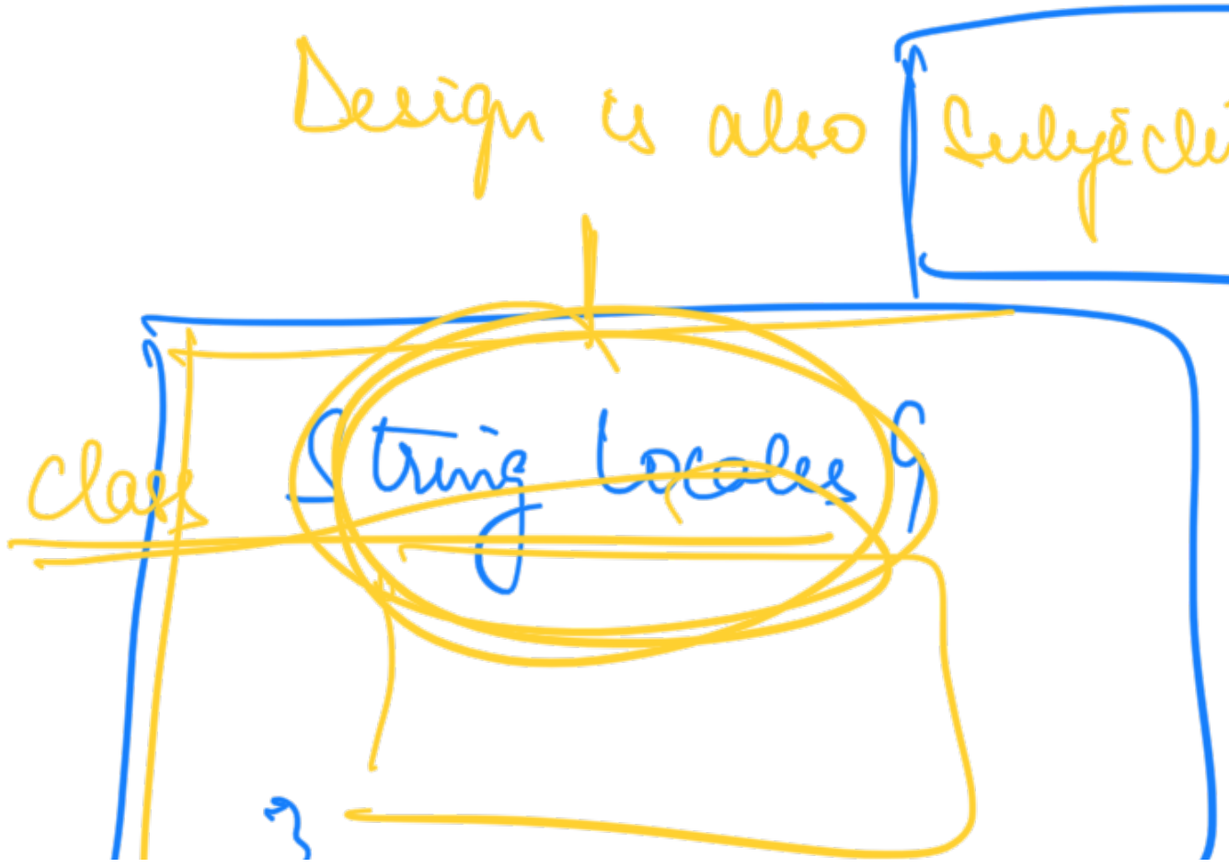
There is no single correct answer in software

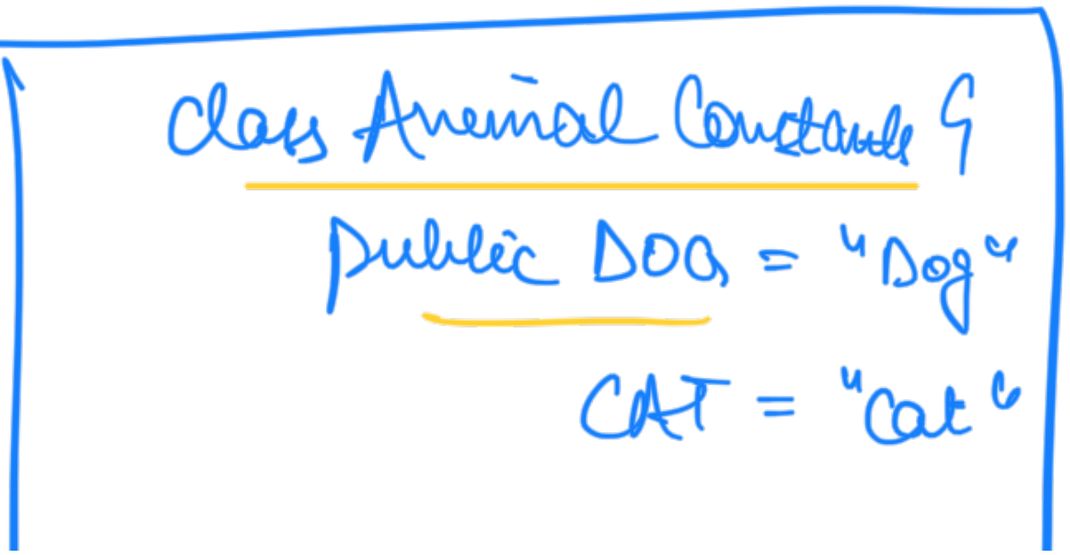
design.

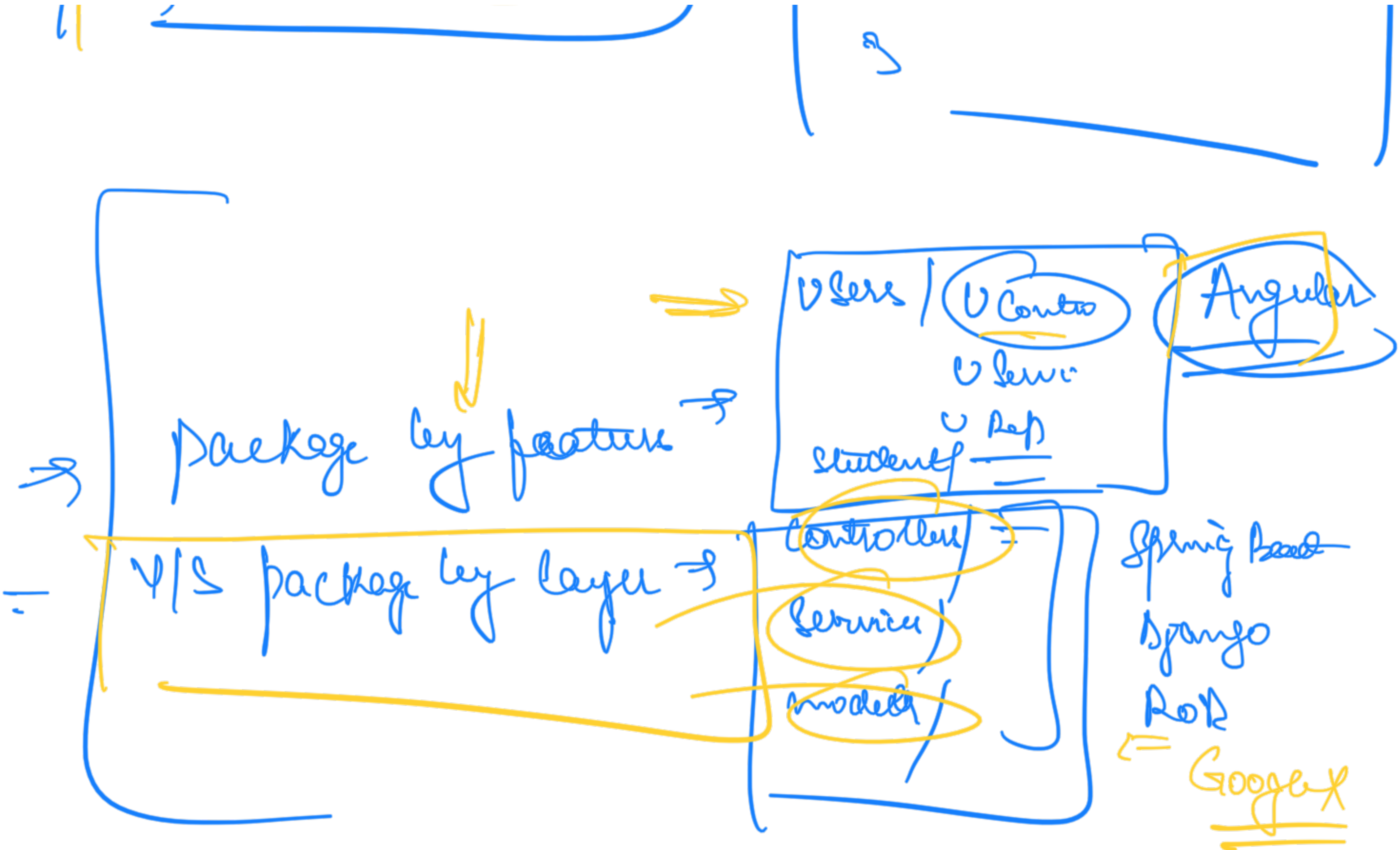
Every action has a tradeoff

→ Following any design principle to 100%. Depth is not practical.

Design is also subjective






Users
U Contro
U Servi
U Rep
Students
Angular
Package by feature
V/S Package by layer
Controller
Service
models
Spring Boot
Django
RoR
Google

SRP Summary

1. Every code unit ⇒ Single Responsibility

2. When is SRP violated:
   - a.) if-else
   - b.) Monster method ⇒ Break into Smaller Methods
   - c.) util/common/helper packages/classes ⇒ Name and Scope them better

3. SRP is Subjective

VO

Build class is also violating ② as …

Bike class is also violating (O) of SOLID

O → Open Close Principle

O → My codebase should be open for extension

but close for modification

Extensibility → adding new feature

Maintainability → ensuring the system keeps on working well even when no

0

new requirement

y

My codebase should make it easy to add new feature but adding new feature should require very less ideally none change in my already written codebase

⇒ Adding new feature should mostly be adding new classes / methods.

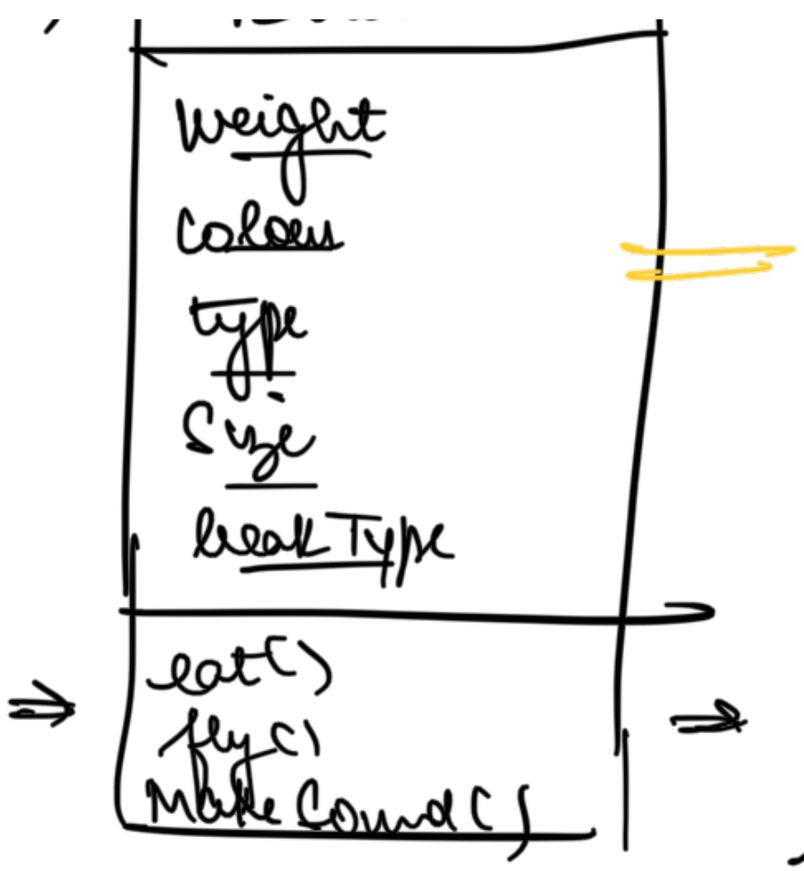
⇒ (1) Testing (No changes in Already existing TC) → Something that used to work earlier stops working

(2) No Regression

SRP and O/C Principle often go hand in hand

(VI)

→ Bird ✓



⇒ ⇒ Methods are there but implementation is not possible

ABSTRACT Classes

⇒ Classes of whom you cannot create an Object

Bird b = new Bird(); ✗

① When you don't want to create object of a particular class

↳ You are only going to create objects of its child class.

Where parent class is just like a stamp (you

User u = new Student();

TA()

want to hold

Objects of child classes

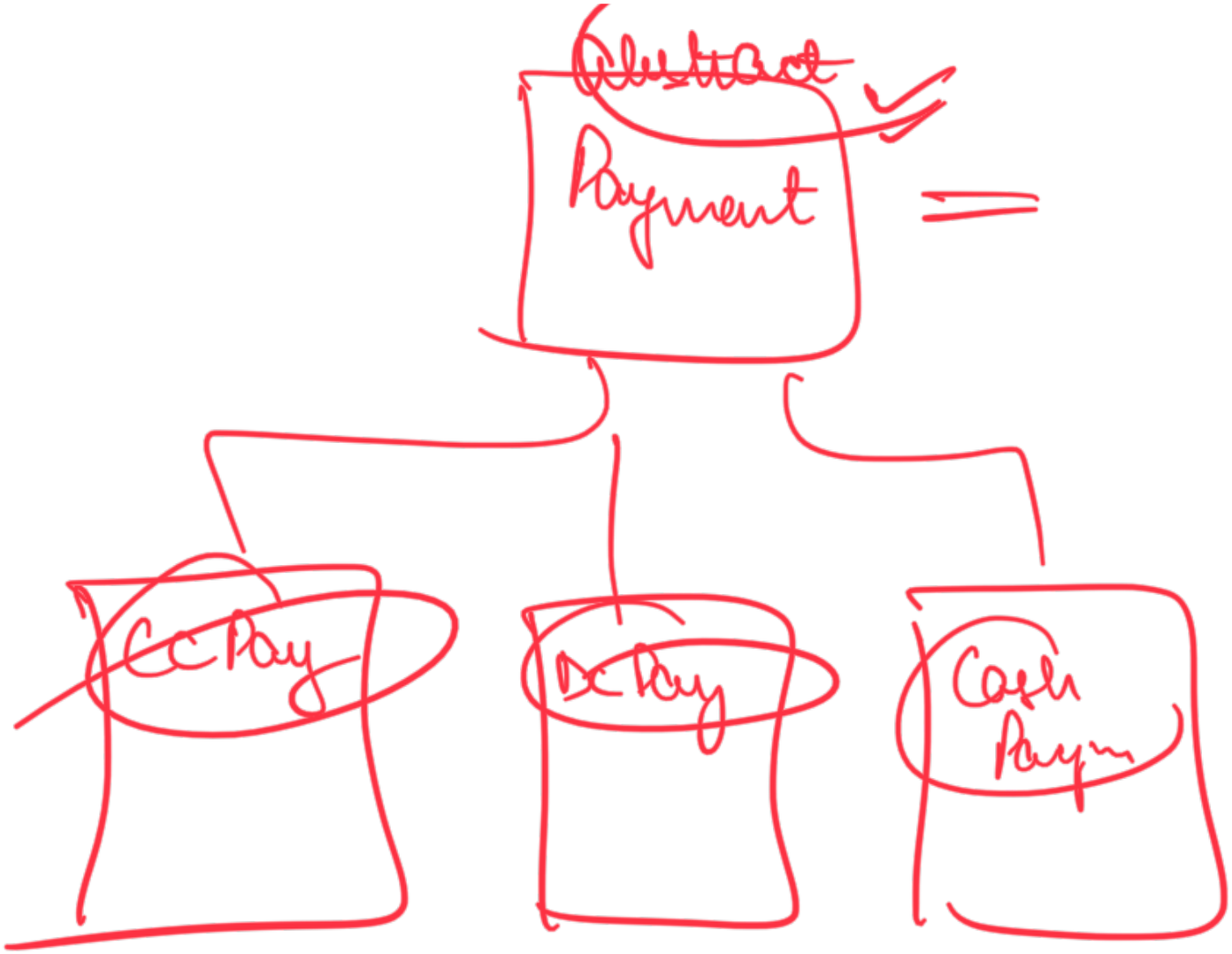
in a variable of parent class)

Mentor()

User u = new User();

User u = new Mentor();

List<User> users = List.of(new Student(),
new TA());



Why Not objects of a class

① Makes no logical sense

(2) When we don't want to implement atleast one of the methods in the class and want to force our child classes to implement them.

Abs class	V/S	Interfaces
→ Attributes		→ No attributes (C.
→ Only 1 extended		→ Any # of interfaces
→ Entities (Physical existence)		→ Behaviour

Bird

Runner

runCy

If attrs + behaviour:

classes

else if only behaviour..

interfaces

V1

Abstract Bird

weight

colour

type

Common attr

Swing

type

V6

Attrs that are common to every Bird

if / else

100 classes

Eye

breakType

Abstract

fly();

eat();

sayCorrectly();

Every Bird flies in a diff way

Every Bird eats and lays in same way

Specific type of Bird

Pigeon

flyers

Crow

flyers

Hen

flyers

Sparrow

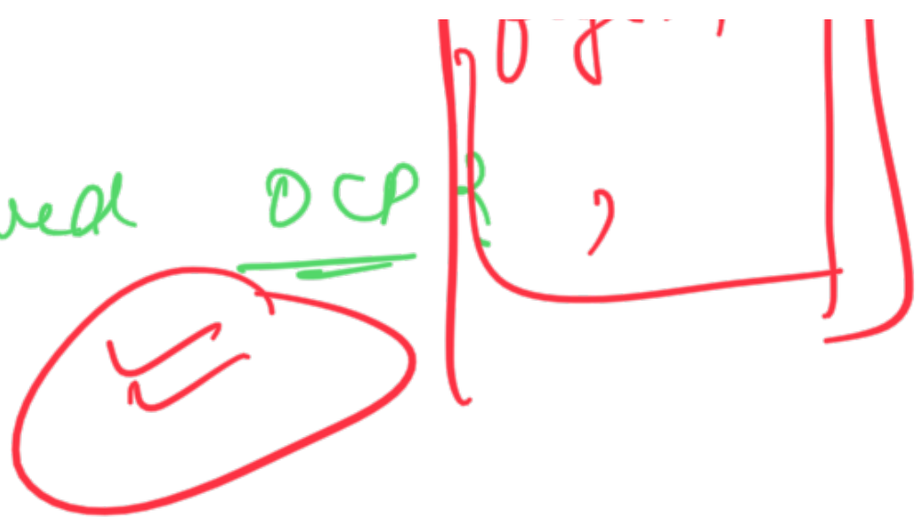
flyers

Owl

flyers

Have we solved SRP

Have we solved OCP ?



when you follow SRP, you end up creating a lot of classes

abstract fly();

Can't fly

Pigeon
fly()

Crow
fly()

Hen
fly()

Sparrow
fly()

Penguin
fly()

① Leave fly blank

② Throw Exception → Penguin can't fly

(L) Liskov's Substitution Principle

NO SPECIAL TREATMENT FOR ANY CHILD

Objects of any child class should be able to be substituted stored in a variable of a parent class without requiring change in further code.

Bird b = new Pigeon(); ⇐ Polymorphism

void doSomething() {

new Penguin

new Crow

new Owl

Bird b = new Pigeon();

b.fly();

if b is not flying :

Ideal Sol^n

If someone can't fly then should not

D

The fly method in Unix

Penguin.fly(c)

No fly method in birds that can't fly

But fly method in birds that can fly

Assg^n: Refactoring a codebase

MVC

Models/

Controllers/