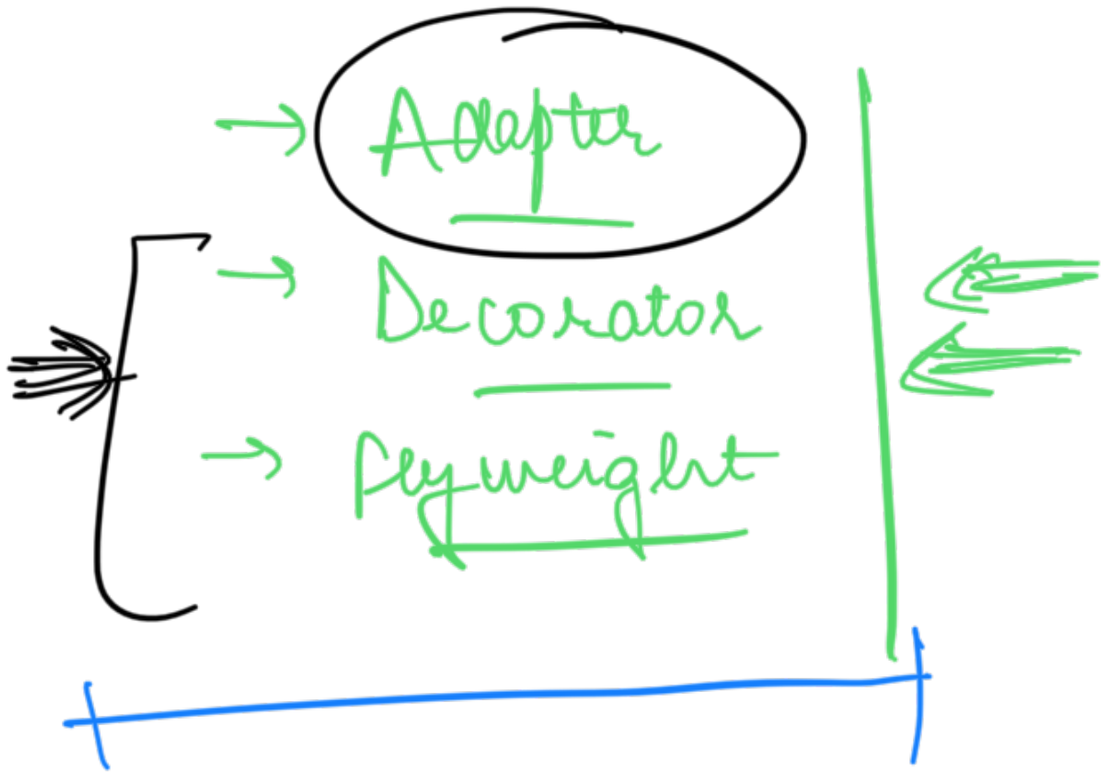
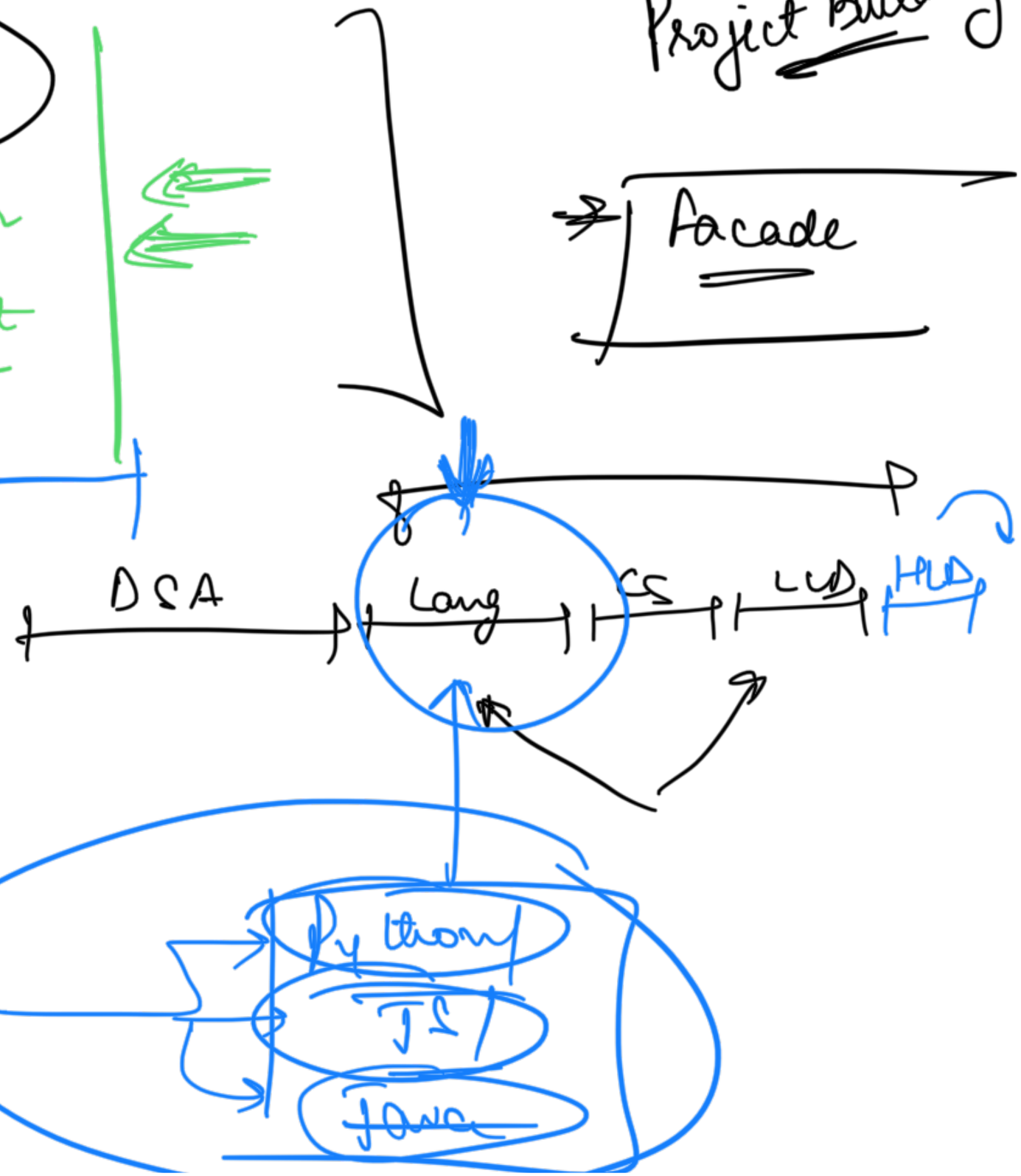
Structural Design Patterns



Schema Design

Project Building



Agenda

Structural Design Patterns

1. Adapter
2. Decorator
3. flyweight

Optional Structural Part - 2

Facade

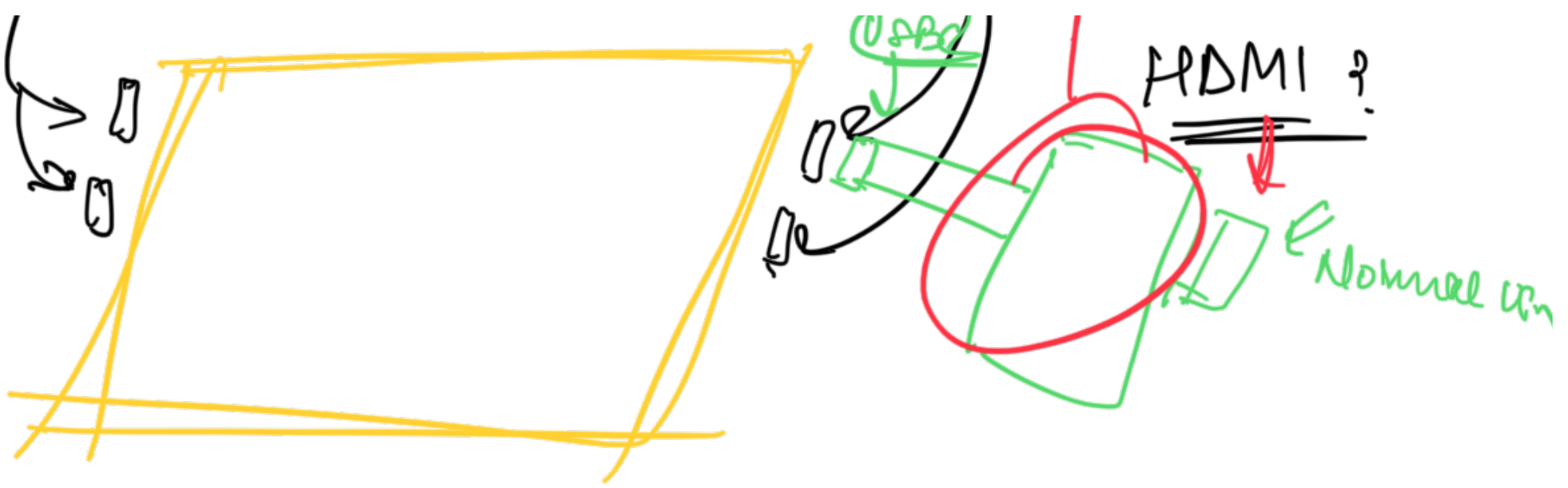

Factory
LD
Intermediate
(Fast)
Tanmay
LD
Beginner
(Slow)
Naman
5 class
10 class
2/3 DP in 1 class
1 → Max
MWF
TTS
MWF

Structural Design Patterns

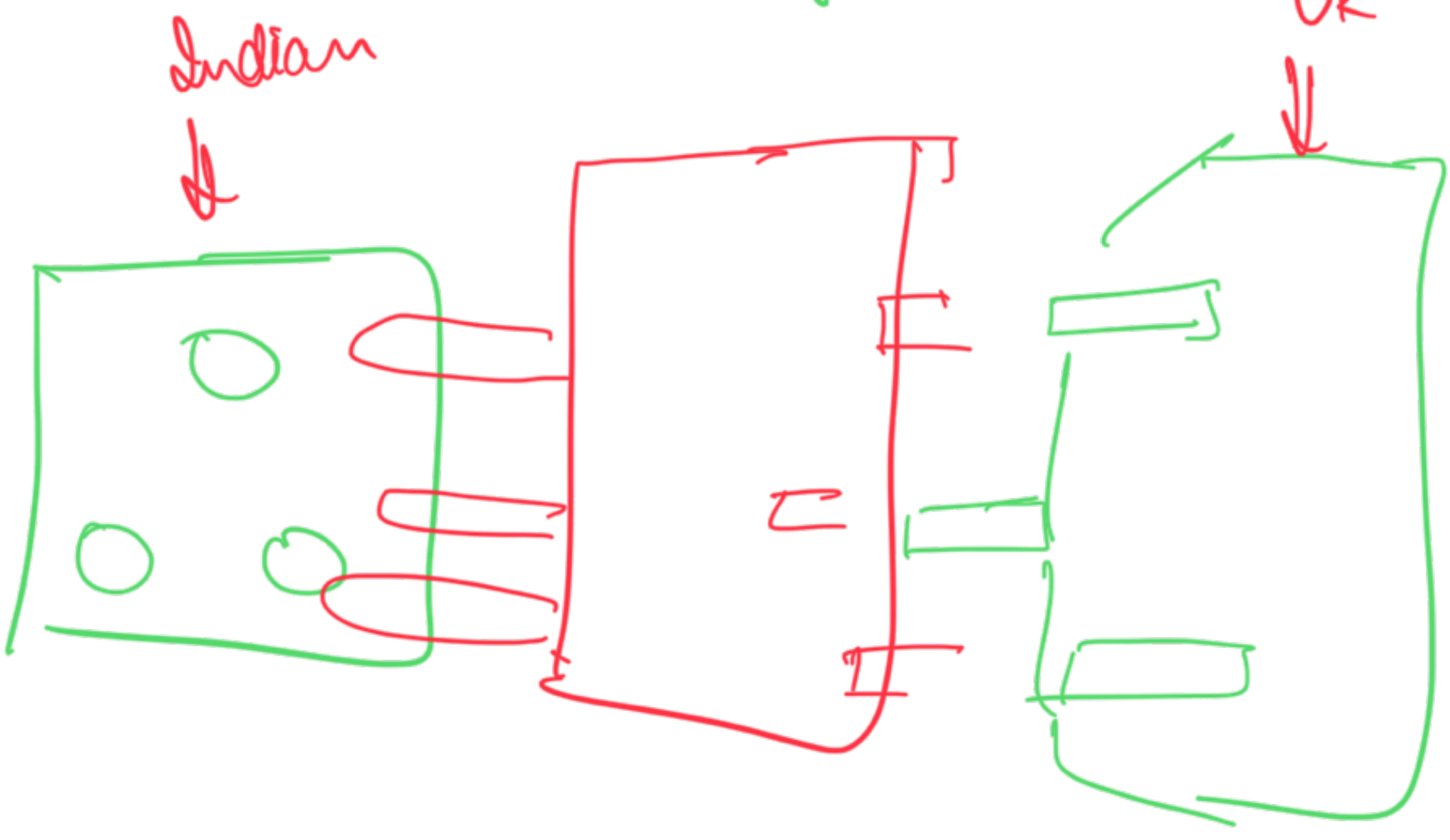
→ How will My code / be structured classes

- Singleton
- Builder
- Prototype
- Factory

Adapter Design Pattern


USB
HDMI ?

A dapter allows to connect a device not directly supported by the hardware



Adapter converts an interface that is not

Supported to an interface that is supported

Software Engineering

Flipkart

Payments Team

Use Payment Gateway

- Billdesk
- Razorpay
- PayU Money
- ccAvenue

makeCCPayment

{

long

}

(CC-No, cvv, exp-date)

String ccNo = CC-No. to String()

String cvv = CC-No. to String()

String expdate = CC-No. to exp date

tID = sp.payVia CC(CCNo, cvv, expDate,)

while (! CheckPayment Success(tID))

wait()

Razorpay

{

payVia CC(String cardNumber, String expdate)

String cvv

checkPaymentSuccess (String transactionID

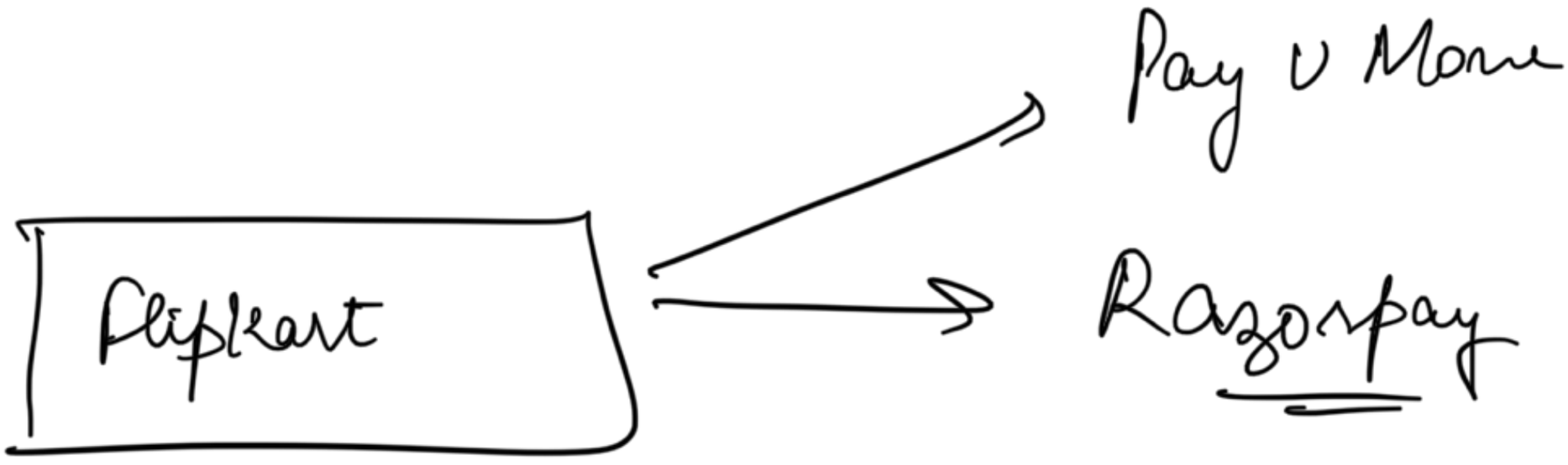
}

PayUMoney {

payByCC (int cardNumber, int expDate,
String cvv, String Name)

getStatus (long tID);

?



⇒ This is a source of lot of bugs

→ SRP is also getting violated

Pt codebase is also concerned with how 3rd party API

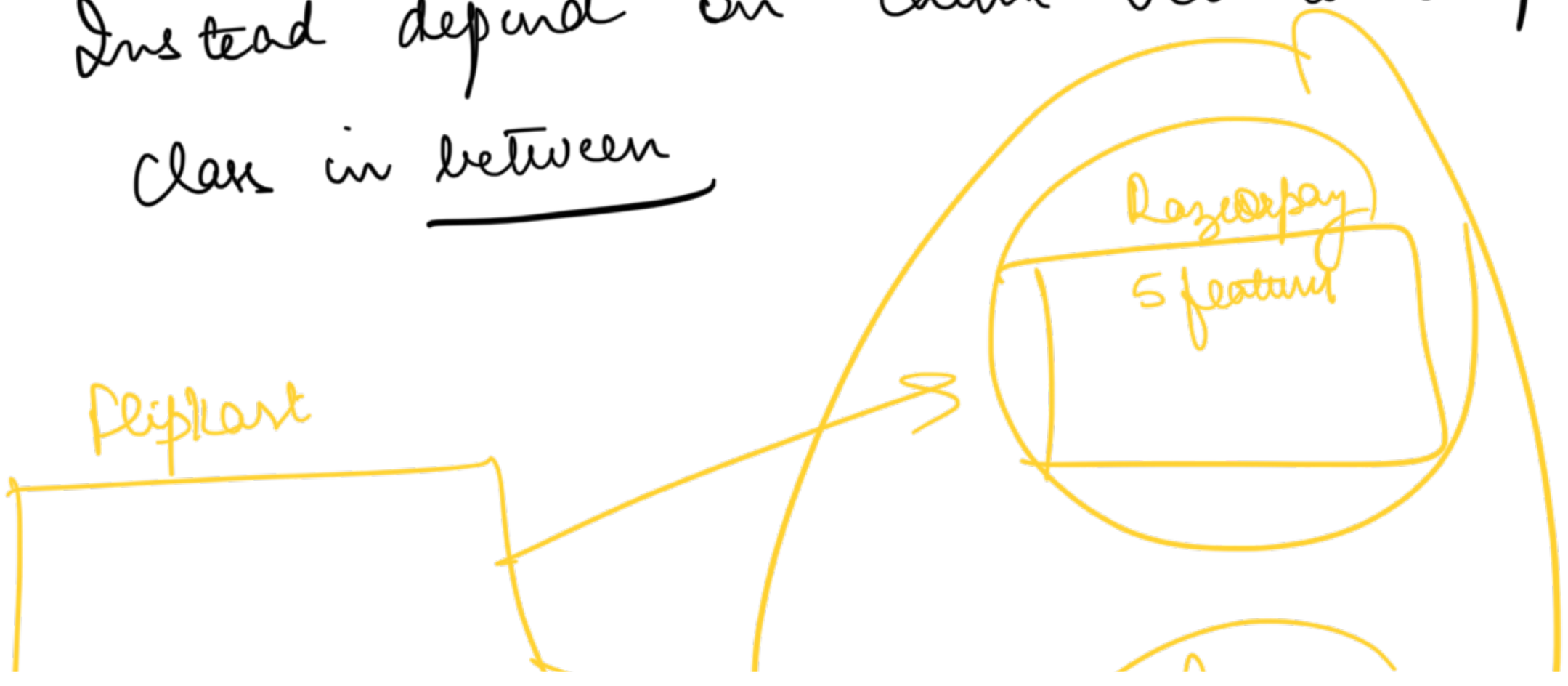
→ Also violates O|C

→ If I change the providers for payment, I will have to open Pt code

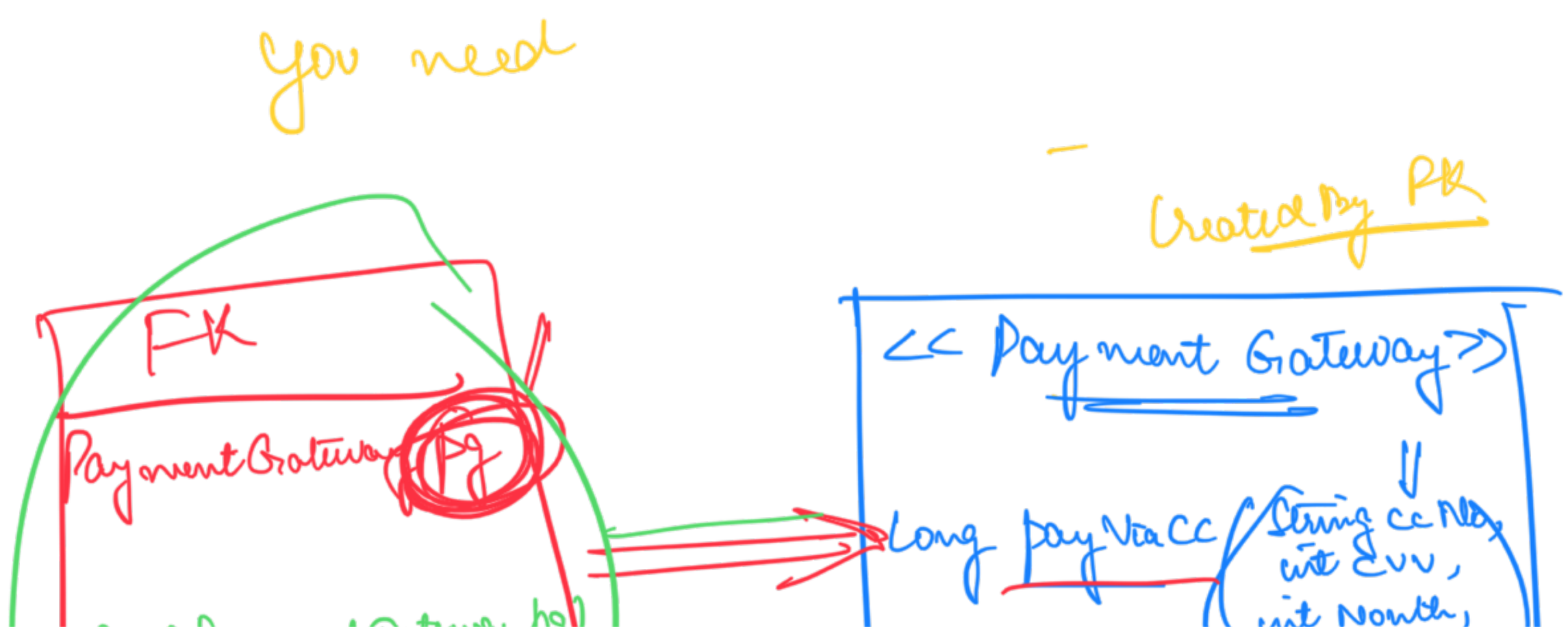
Adapter design pattern

When you are using 3rd party APIs that are prone to change, don't depend on them directly:

Instead depend on them via an adapter class in between



① Create an interface for the services those you need



fk (Payment Status) pg

this - pg = pg

Payment Status

get Status (long id)

city year

O/C

SRP

DI

Create

RazorpayAdapter

Razorpay sp

pay via CC ( )

sp-pay via CC

Razorpay

BilldeskAdapter

Billdesk

Adapter DP

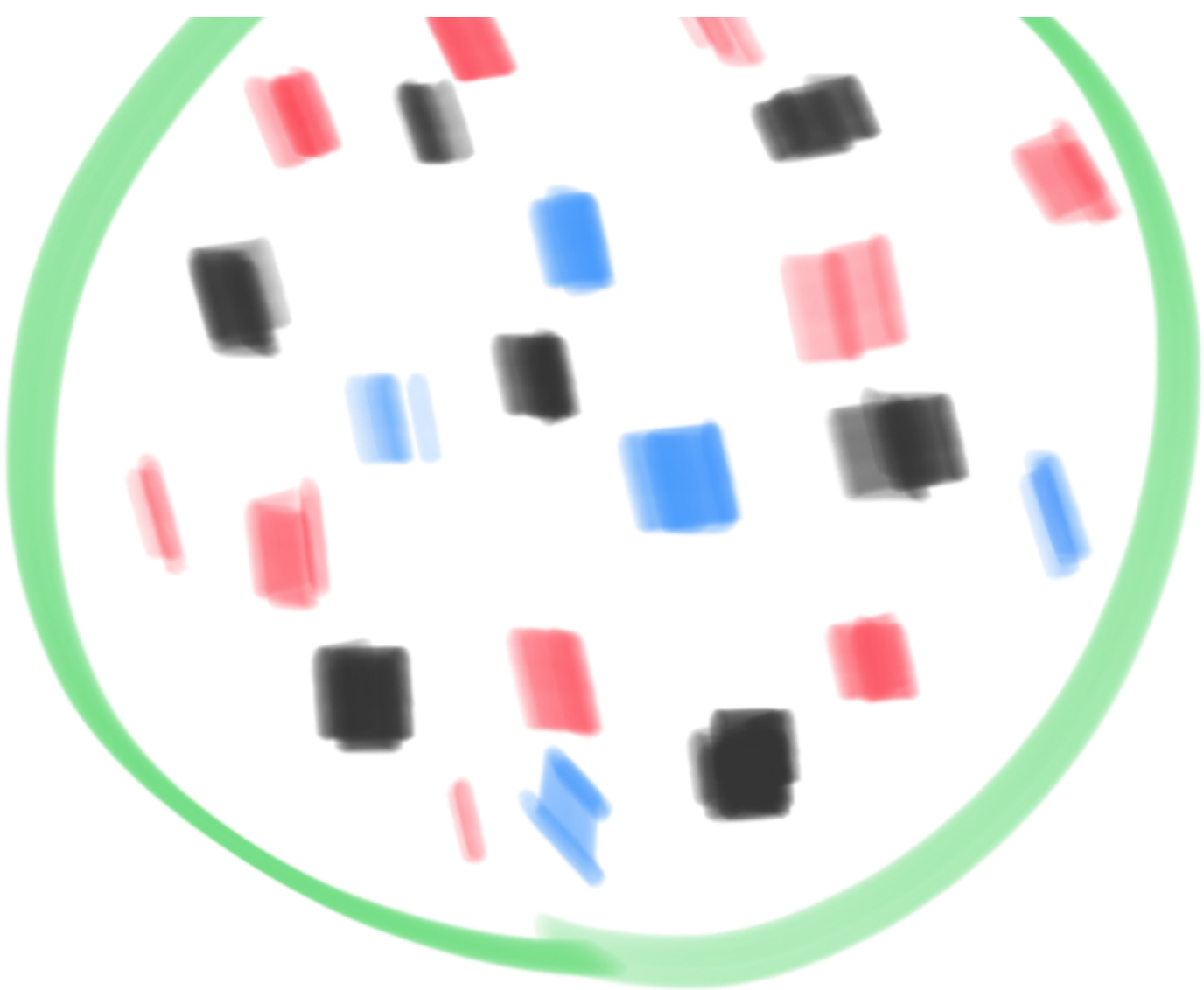
when your codebase needs to interact with a third party API, don't call the codebase of 3rd party API directly

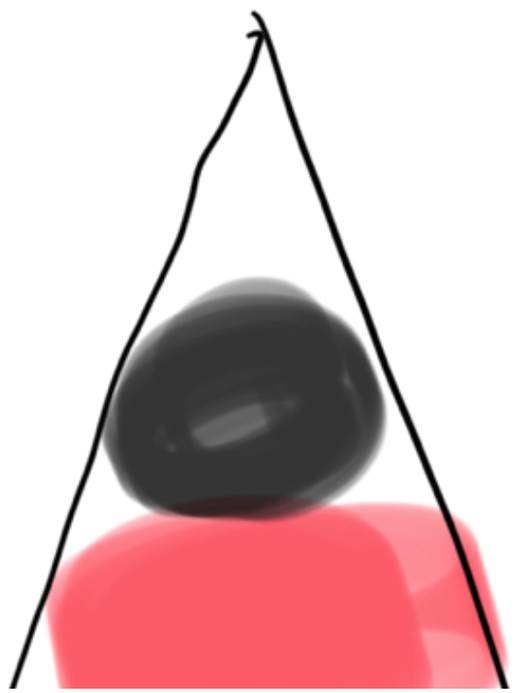
① Create an interface

② Implement an adapter class for each variant of 3rd party API

Pay U Money

Decorator Design Pattern





Decorate ?

Gift

By decorating something, the category of the object I am decorating still remains the same but some new features are added/ removed. Like it might change price etc.

Which

Domino's Pizza Ordering System

interface Pizza {

→ get Cost();

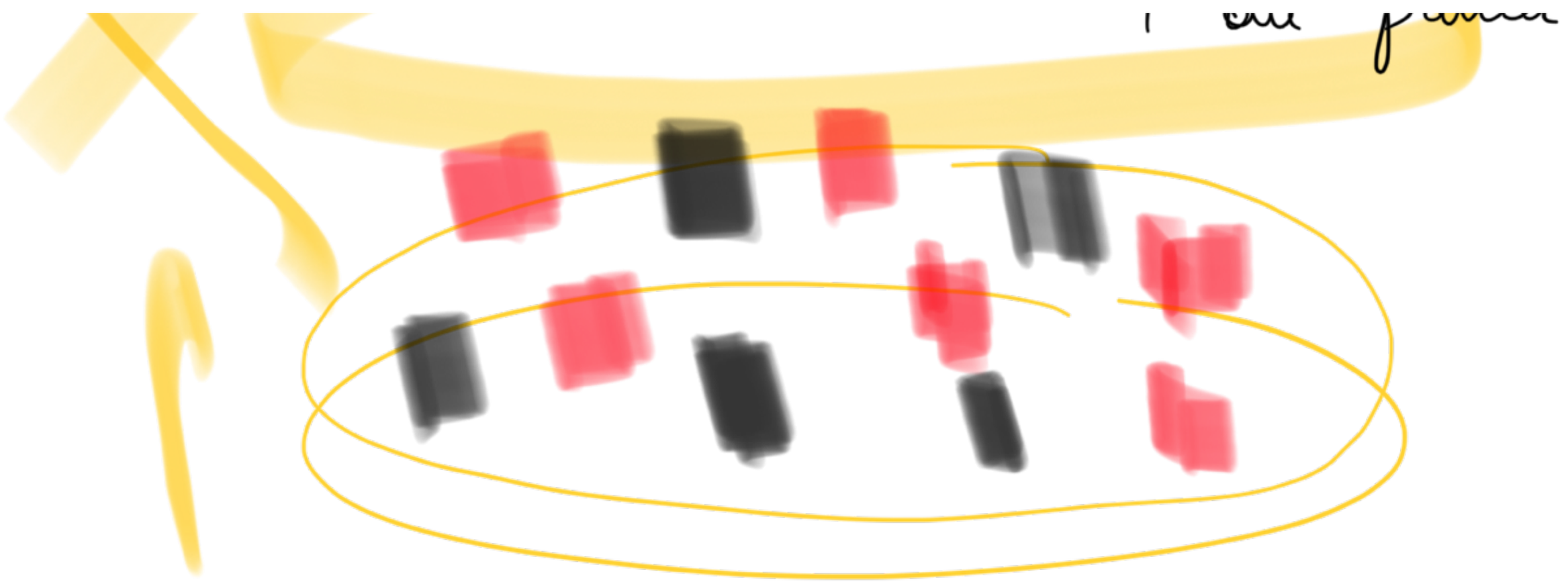
→ get Constituents()

}

One base + One base +

One topping + One cheese

+ Dough



Question Implement a Pizza Creator

~~List<Topping> toppings;~~

List<Item> items

→ addBase(Base b)

items.add(b)

addTopping(Topping)

items.add(topping)

Problem Statement

I want to allow creation of a Pizza

→ (1) either from Scratch

(2) from an already known Pizza

Margherita + Cheese + Chicken + Tomato

Base int

Topping int

Generic Pizza {

addBase()

addToppings()

}

Margherita {

List<Base>

List<Topping>

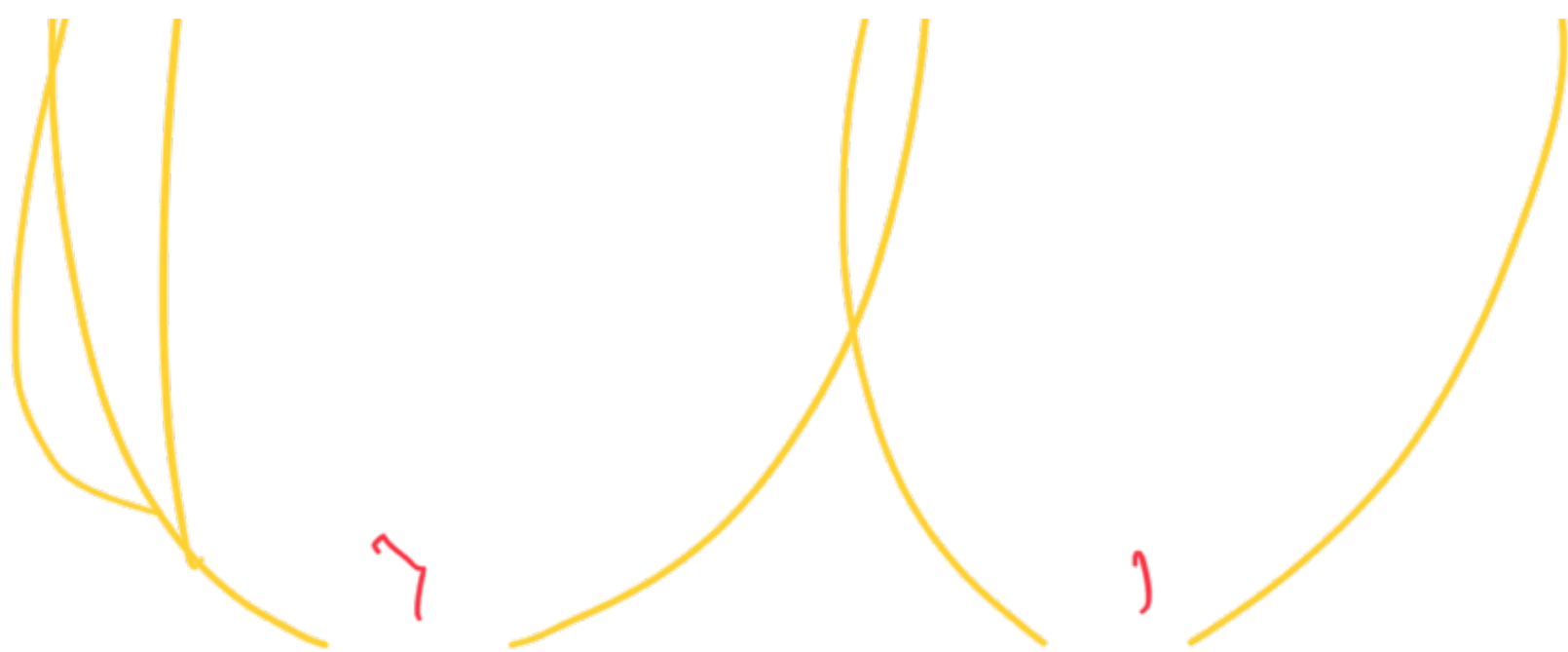
}

Farmhouse {

List<Base>

List<Top

}



Ma

Pizza {

list <Base>

list <Topping> =

addTopping() {

① You have a type of objects to create

interface Pizza {

getCost()

getConstituents()

}

② Decorating == adding feature

before D decorated ⇒ Pizza

after D decorated ⇒ Pizza

We will decorate a Pizza by wrapping

it

Let's create the base class

⇒ from where the creation of pizza will start from.

⇒ for every feature that we add, implement a decorator class for that.

class ThinCrust implements Pizza {

→ Pizza inside

ThinCrust (Pizza pizza) {

}

get Cost() {

inside.getCost() + 100)

get Toppings(Constituents)

inside.getToppings() + ThinCrust

```
interface IceCreamCone {

    getCost()

    getConstituents()

}
```

```
class Cone implements IceCreamCo

    getCost() {
        return 10;
    }

    getConstituents() {
```

class Vanilla Scoop implements IceCream Cone {

IceCream Cone cone;

Vanilla Scoop (IceCream Cone cone) {

this.cone = cone

}

getCost() {

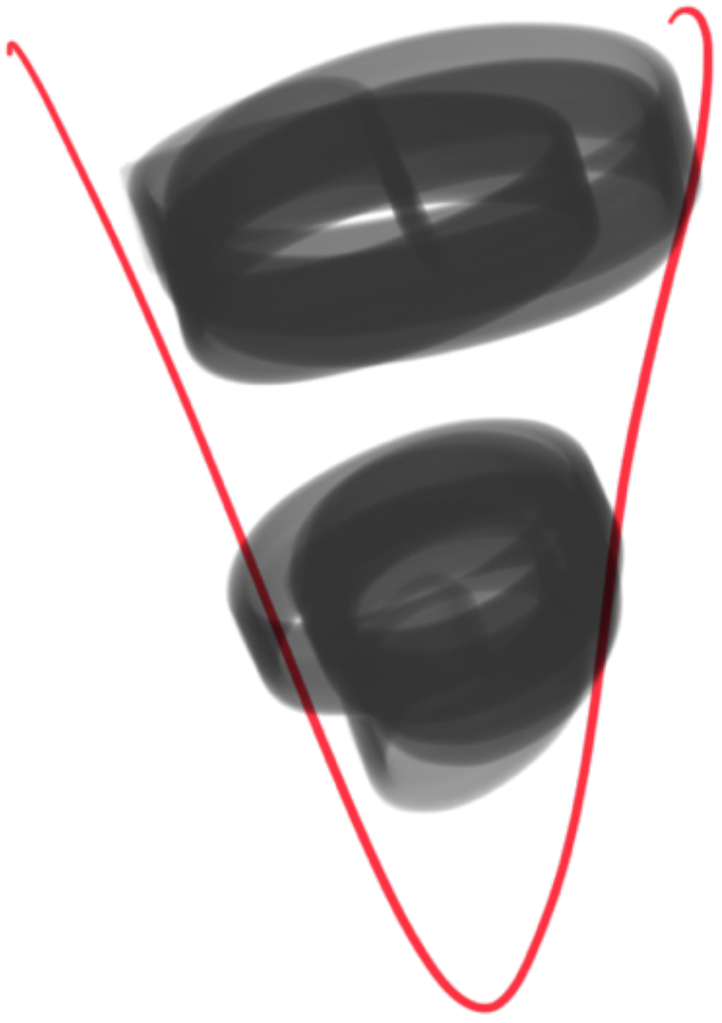
cone.cost + 20;

}

get Cost() {

Cost + Vanilla

}



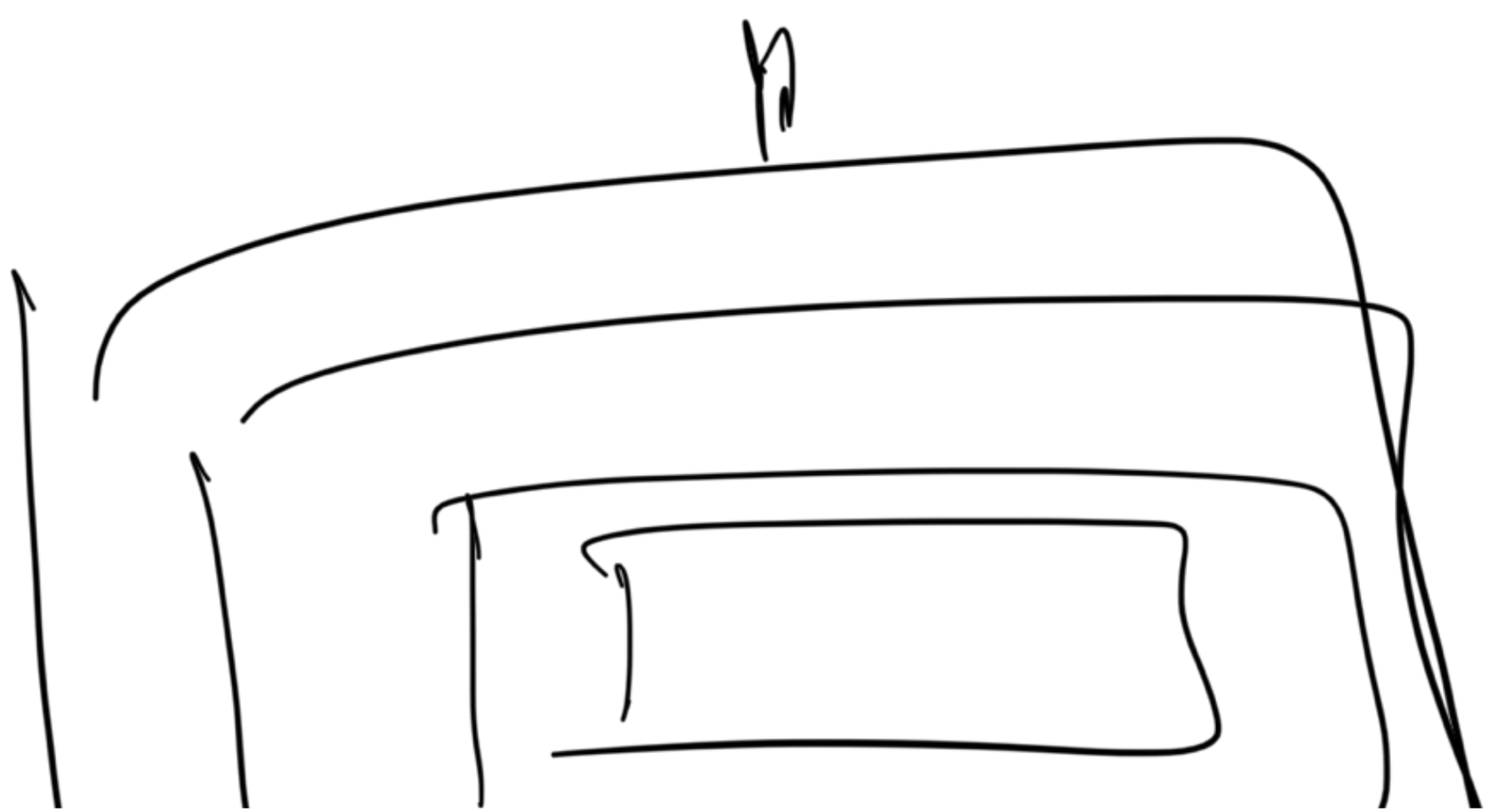
new Pista (

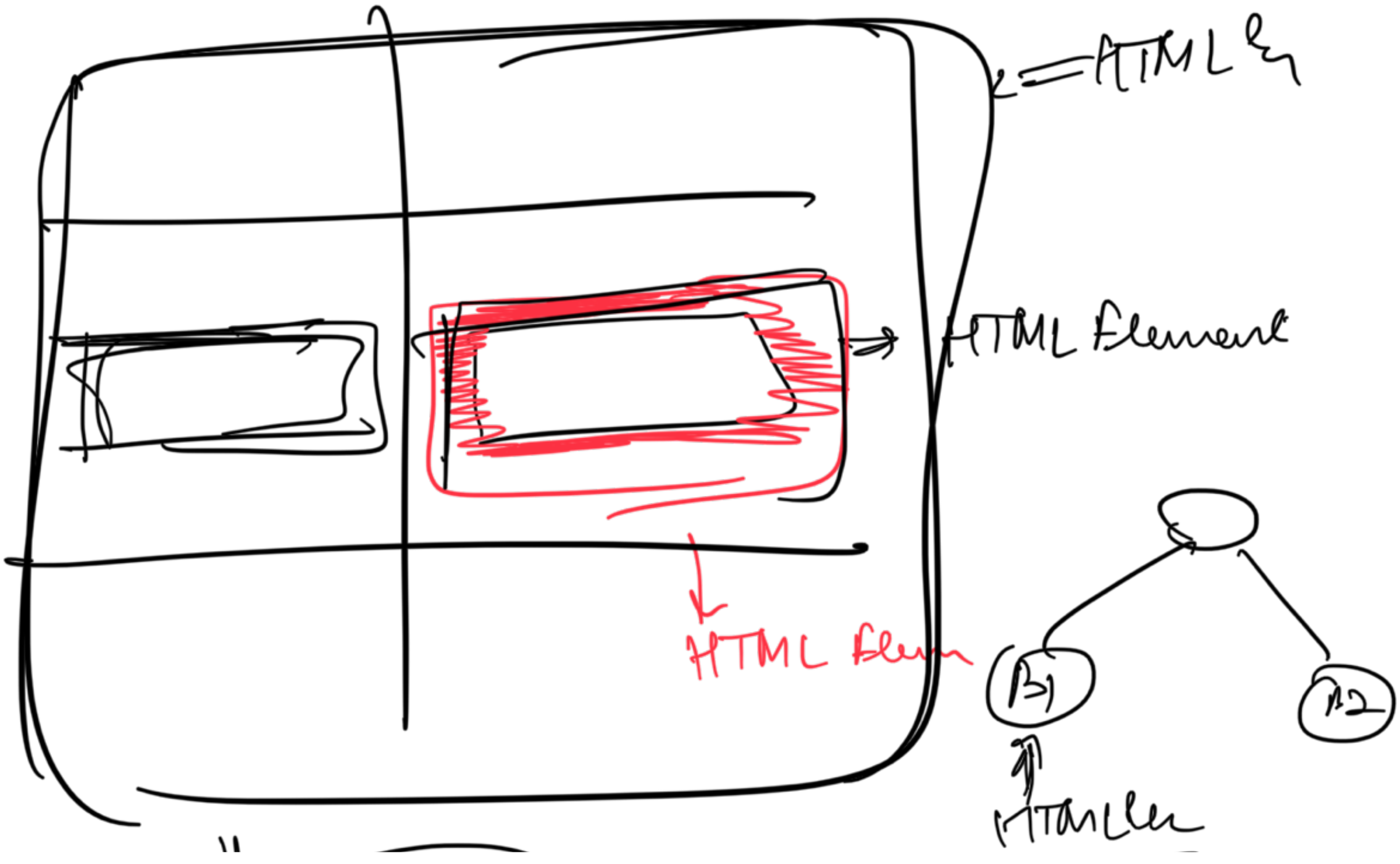
new Butter & cotch Scoop (

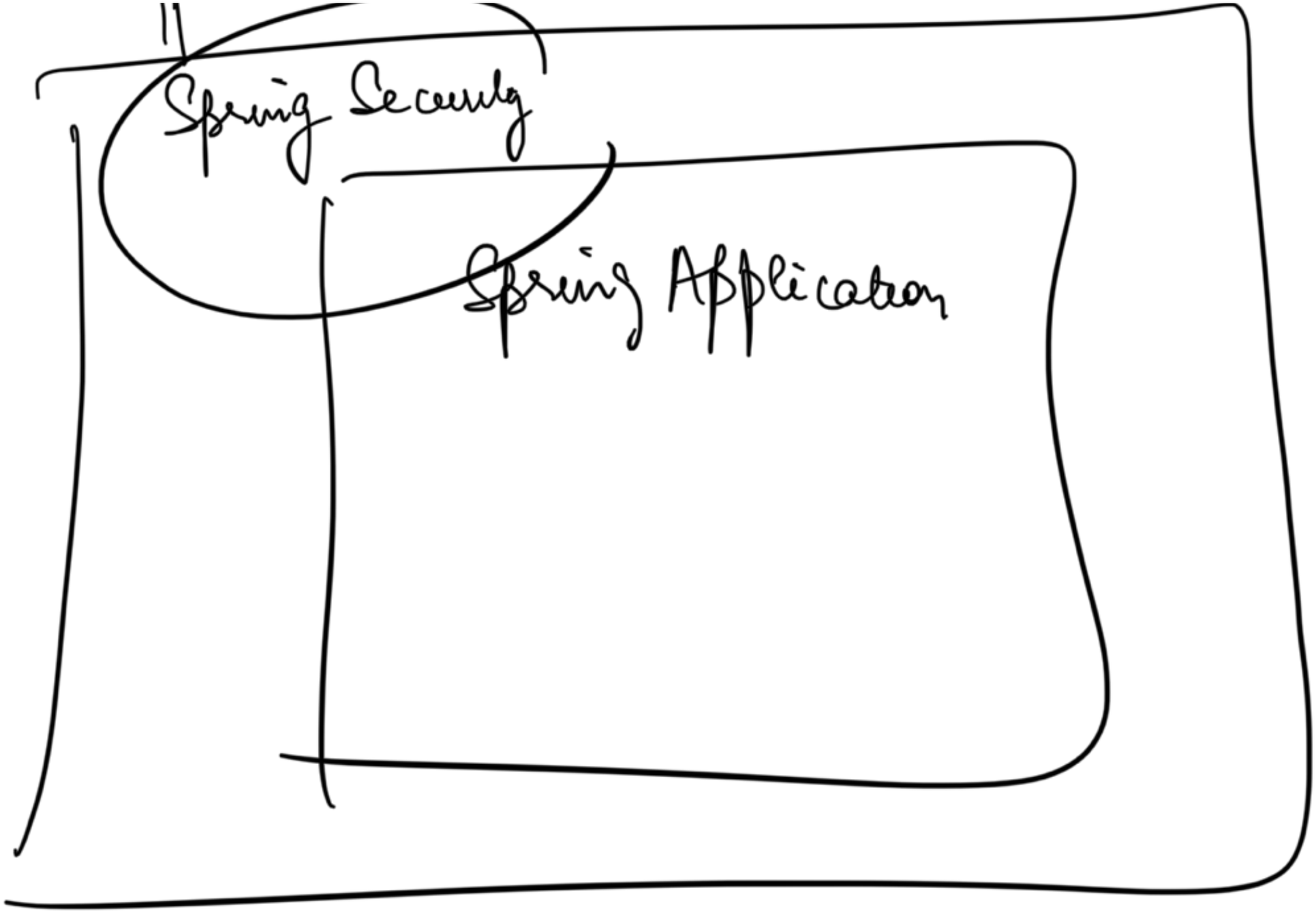
new Vanilla Scoop(

new line ↵

```
new Border(
        new Button(
                new Text ("Hello")
        )
)
```




HTML Element
HTML Element
B1
B2



How is it different from inheritance

W U

Inheritance → Compile time

Decoration → Runtime

Pizza current = —

String newTopping = "";

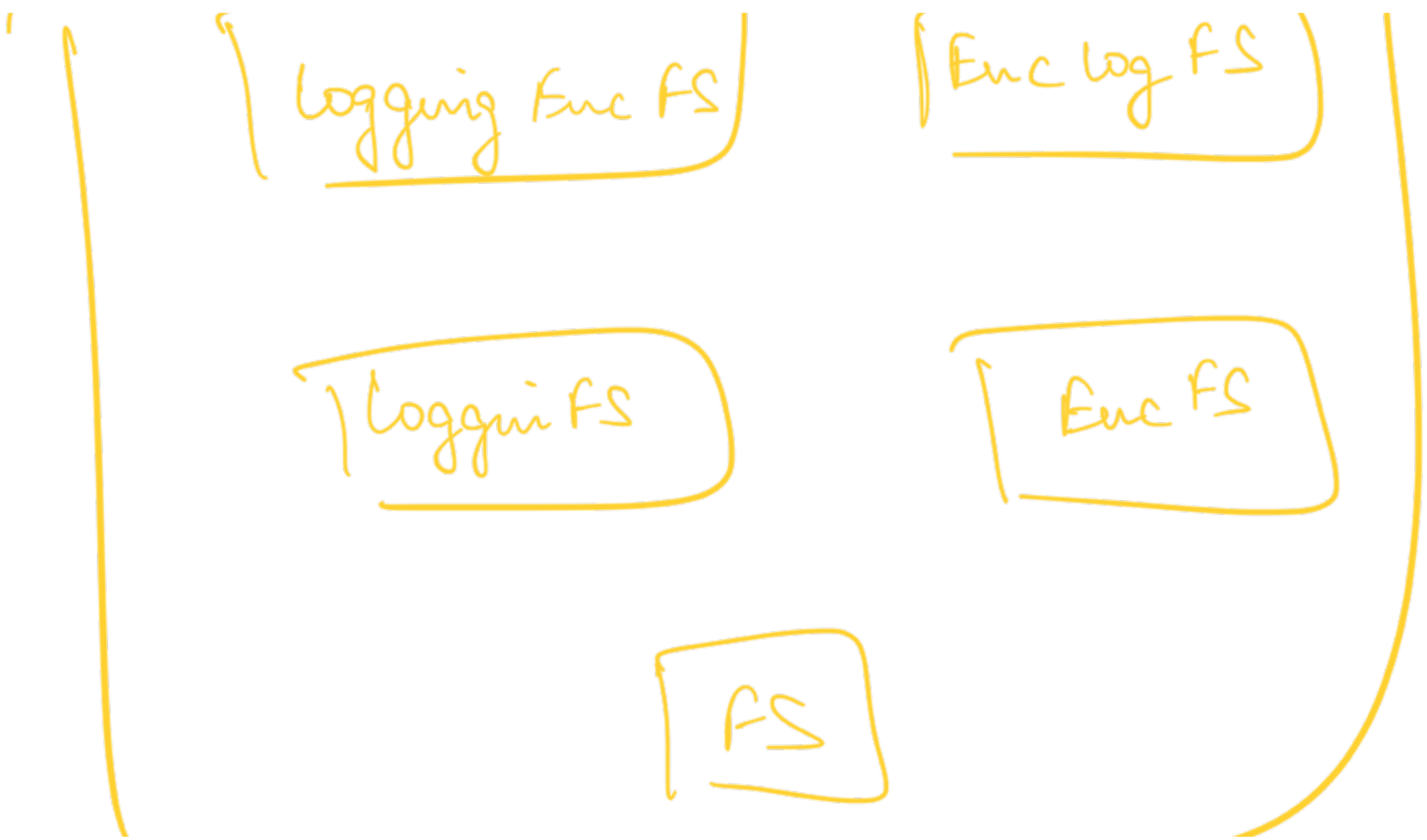
if newTopping == Cheese

current = new Cheese(current)

Decorators

add a functionality to an already existing functionality

you can decorate in any order


loggingEncFS
EncLogFS
LoggiFS
EncFS
FS