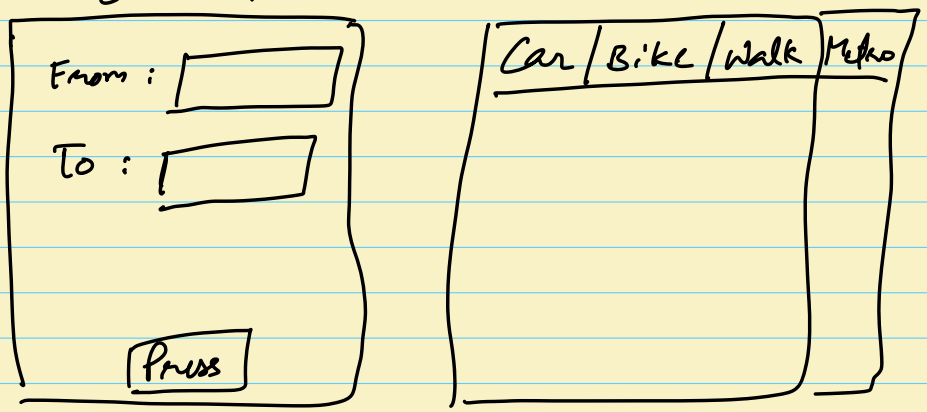
1. Good Evening
2. Lecture begins at 9:10pm
3. Topic - Strategy & Observer

Agenda

1. Strategy Design Pattern
2. Observer Design Pattern

Strategy Design Pattern

Story : Google Maps



v1

class GoogleMaps {

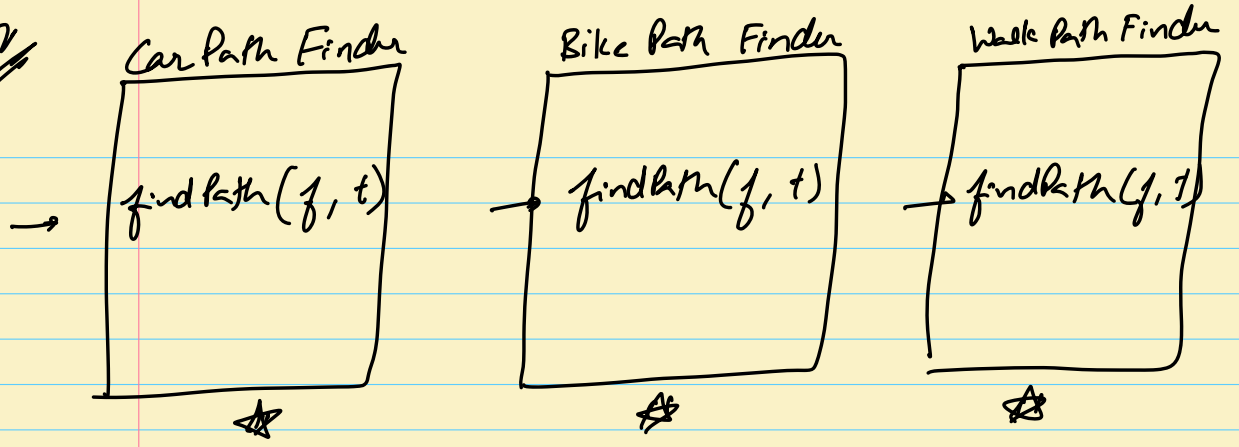
findPath (from, to, mode) {
if (mode == "car") {
// Path via car
}
else if (mode == "bike") {
// Path via bike
}
else if (mode == "walk") {
// Path via walking
}
}

v1 - problems
- SRP
- OCP

violated

Step1 → Create separate class for each mode of travel

V2

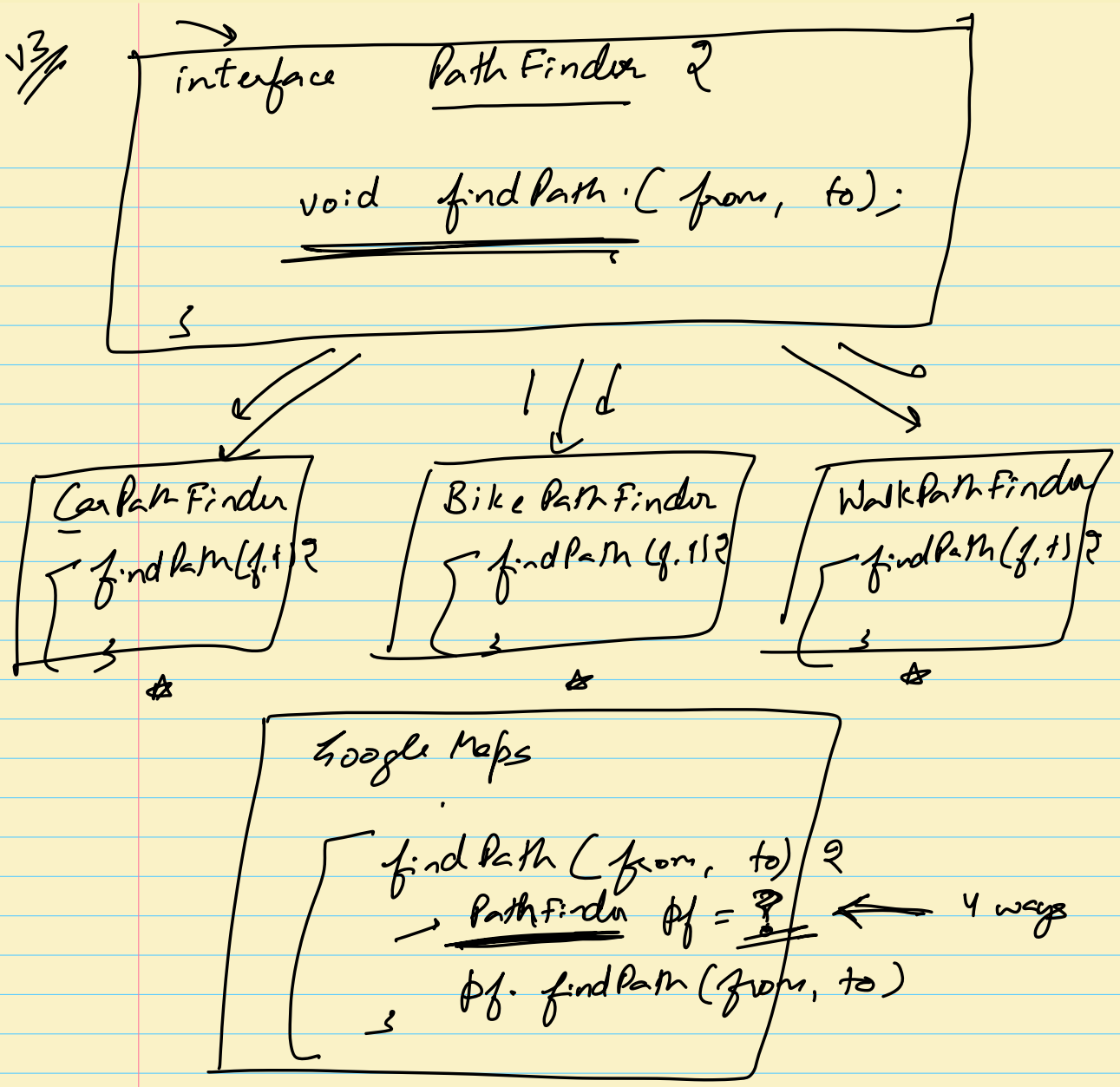


class Google Maps 2

```
void findPath( f, t, mode ) {
    if (mode == "car") {
        CarPathFinder cf = ...
        cf.findPath(f, t)
    }
    else if (mode == "bike")
    ...
}
```

V2 - problems

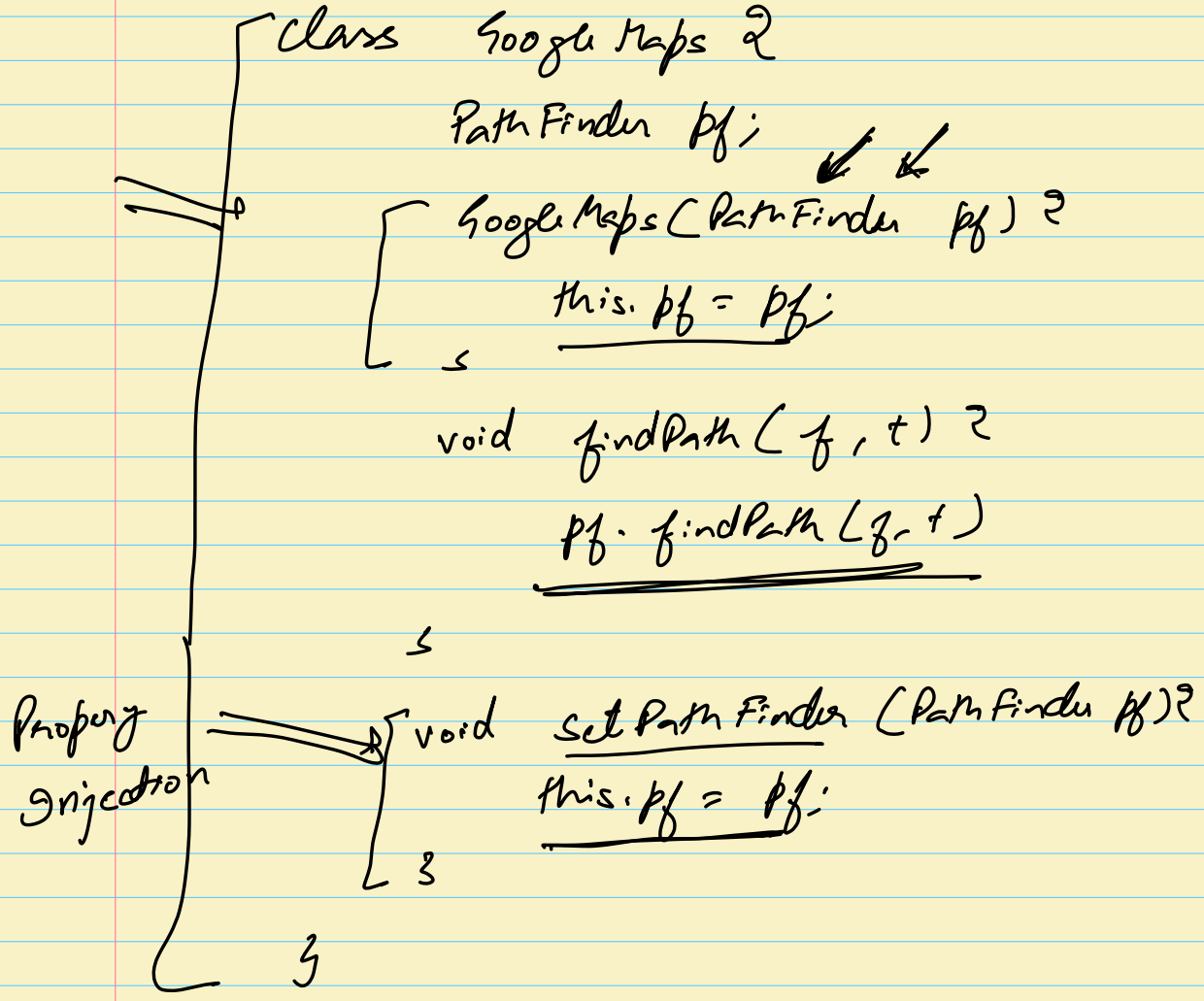
OCP is still violated



4. ways

1. Dependency Injection ✓
4. Factory Method
3. Abstract factory ✓
2. Normal Factory ✓

Way 1 : Dependency Injection



Client's code

```
PathFinder pf = new CarPathFinder()
GoogleMaps gm = new GoogleMaps(pf)
gm.findPath(f, t)
gm.setPathFinder(new WalkPathFinder())
gm.findPath(f, t)
```

Way 2:

PF

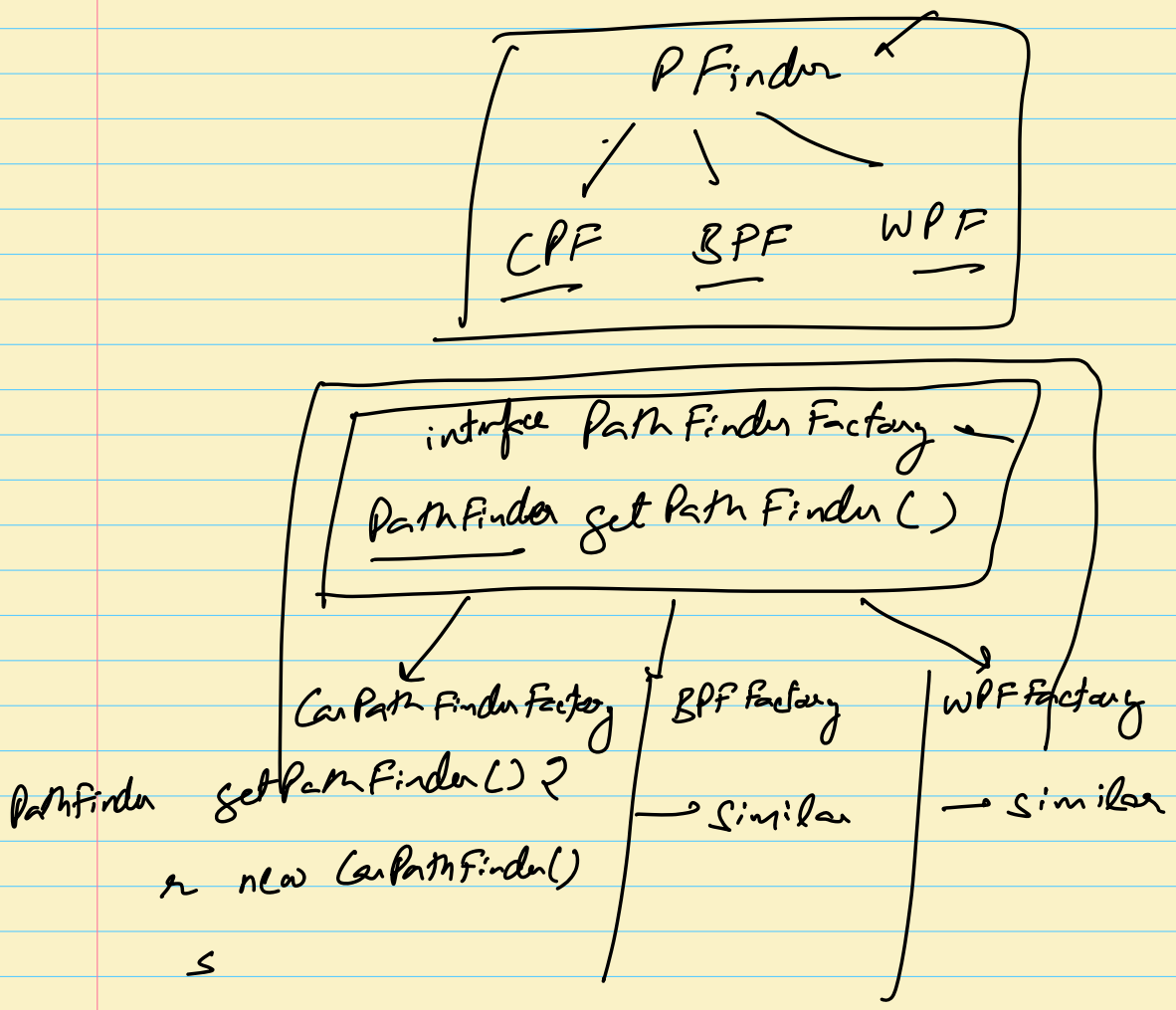
CPF BPF WPF

```
class GoogleMaps {

    void findPath(f, t, mode) {
        PathFinder pf = PathFinderFactory.
                        getPathFinder(mode);
        pf.findPath(f, t);
    }
}
```

```
class PathFinderFactory {

    static PathFinder getPathFinder(mode) {
        if (mode == "car") {
            return new CarPathFinder();
        }
        else if (mode == "bike") {
            ...
        }
    }
}
```

→ SRP is not violated, OCP is but Practical Factory allows it.

Way 3 : Abstract Factory



PathFinder getPathFinder() ?

return new CarPathFinder()

;

class GoogleMaps ?

PathFinderFactory pff;

void setPFF (PathFinderFactory pff)?

this.pff = pff;

;

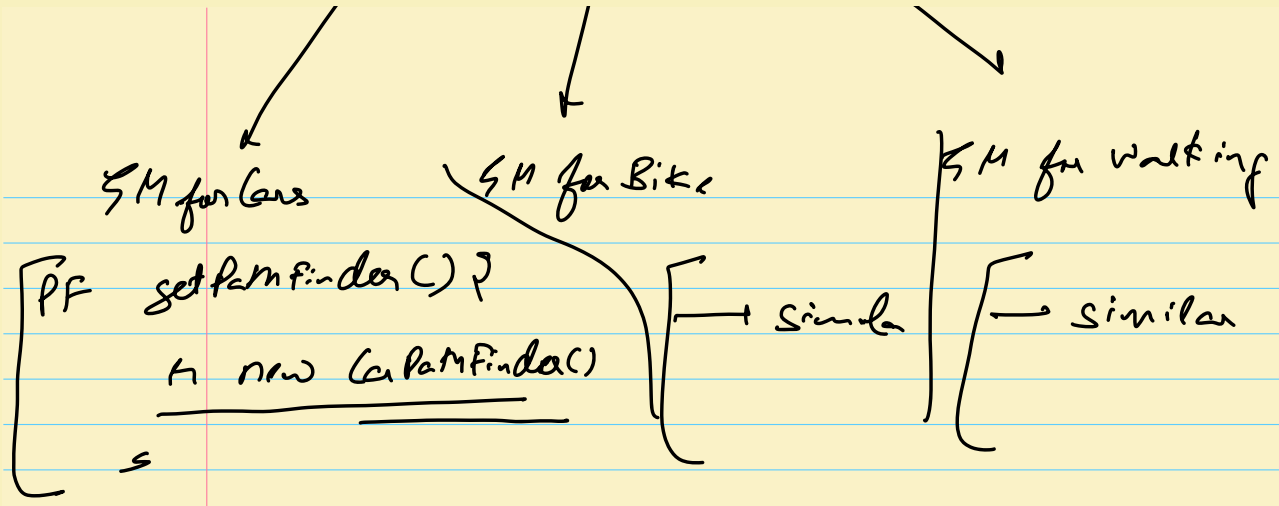
```
void findPath (f, t) {
    PathFinder pf = pff.setPathFinder();
    pf.findPath(f, t)
}
```

Client

```
PFF pff = new CarPathFinderFactory()
GoogleMaps gm = new GoogleMaps()
gm.setPFF(pff)
gm.findPath(f, t)
```

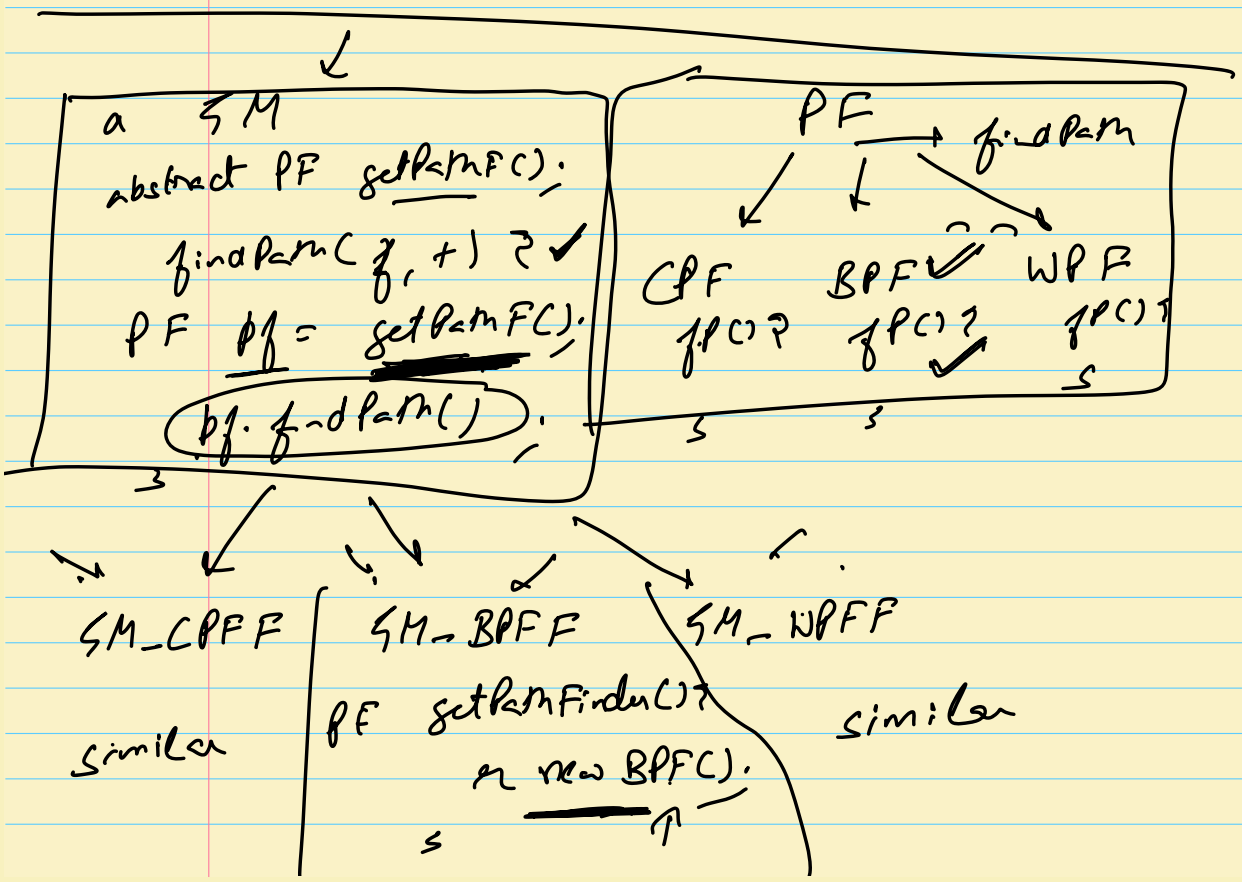
Way 4: Factory Method .. [FM → AF]

```
abstract GoogleMaps {
    abstract PathFinder setPathFinder();
    void findPath (f, t) {
        PathFinder pf = setPathFinder();
        pf.findPath(f, t)
    }
}
```



Client

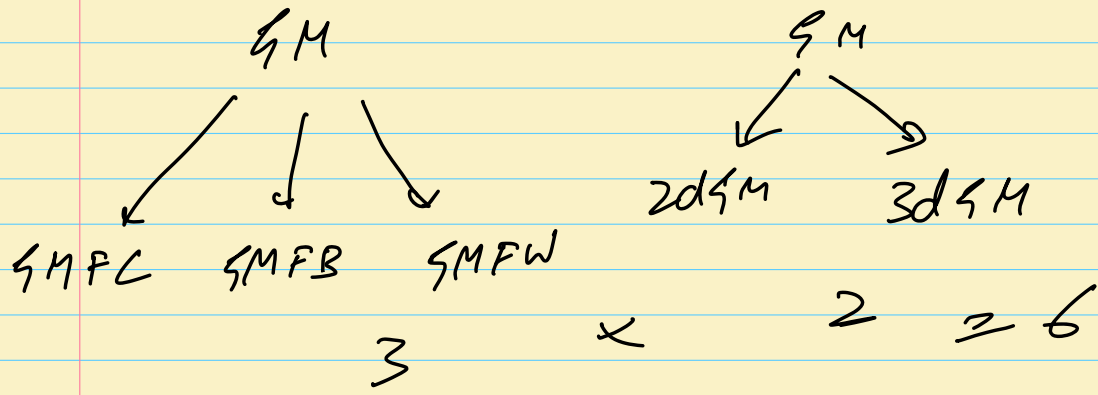
```
GM gm = new GMforCars();
gm.findPath(f, t)
```



Client

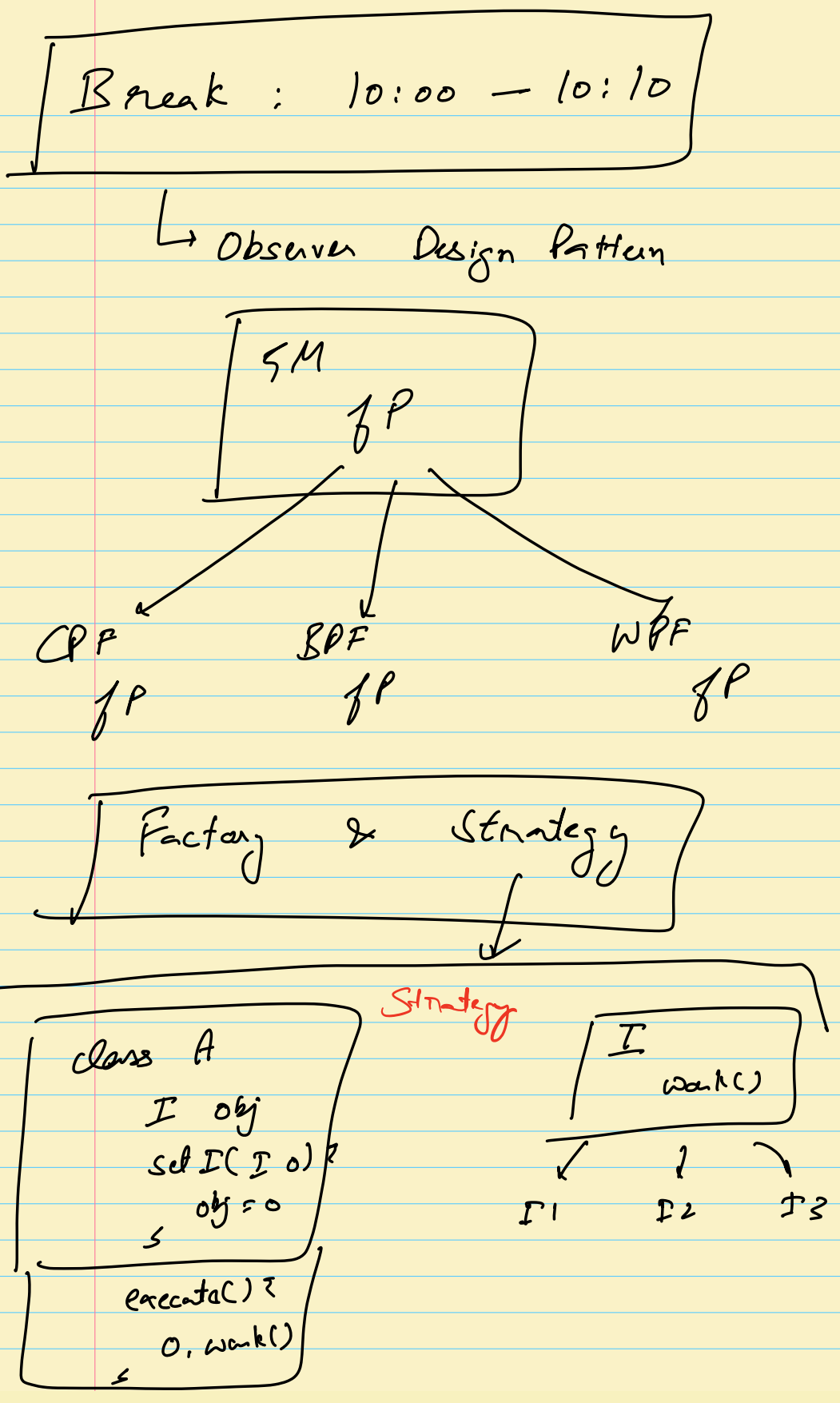
```
GM gm = new GM_BPFFC();
gm.findPath(f, t)
```

Way 4 has problems → Prefer DO or NF or AF



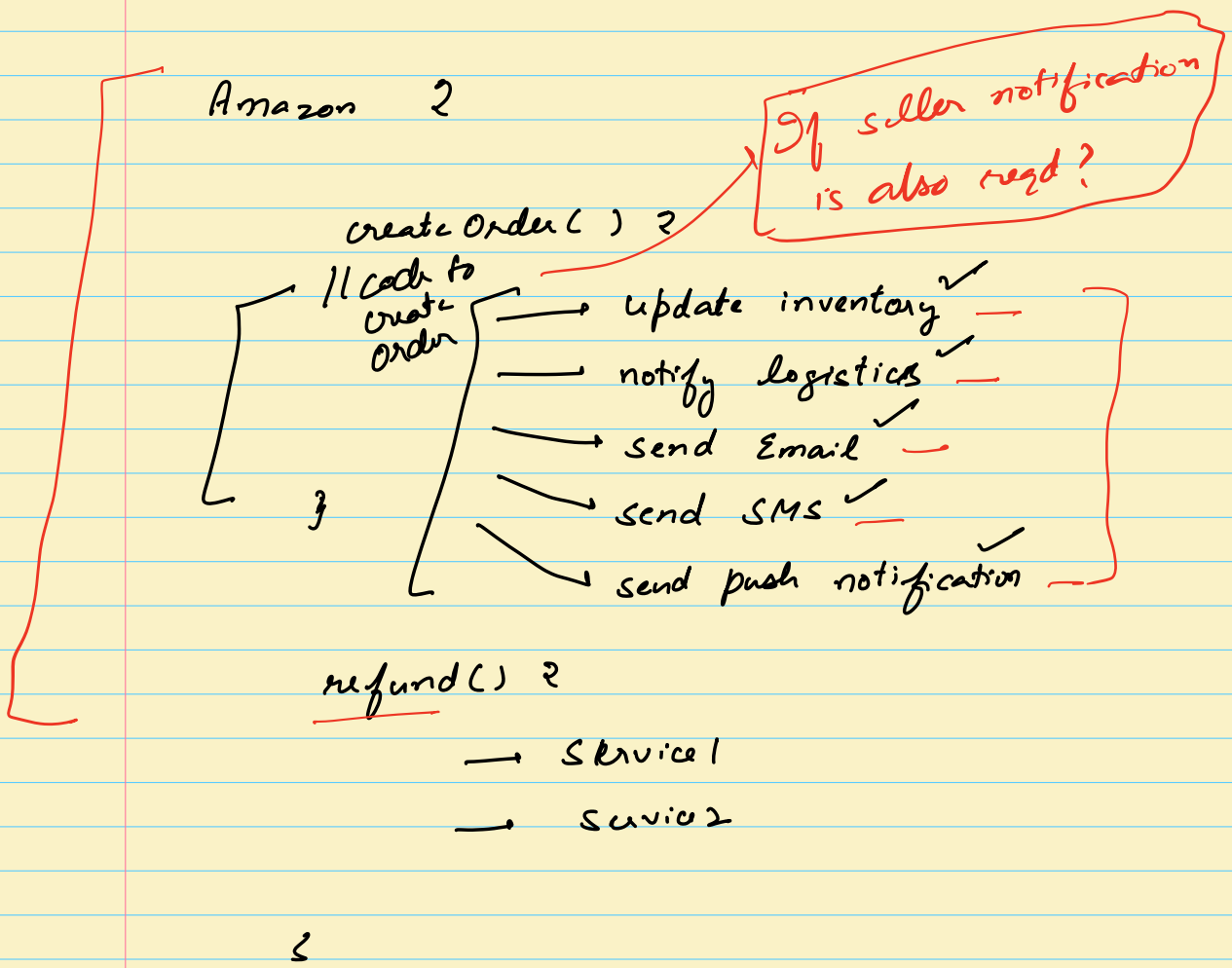
Homework → Code the story discussed above.

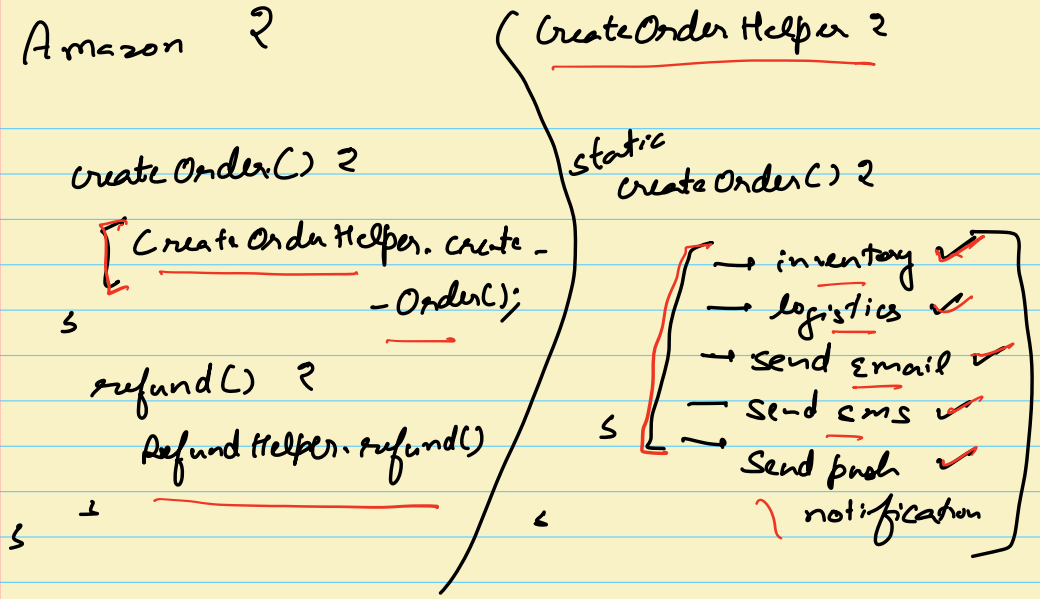
Bird story


Break : 10:00 — 10:10
Observer Design Pattern
SM
fP
CPF
fP
BPF
fP
WPF
fP
Factory & Strategy
Strategy
class A
I obj
setI(I o){
obj = o
}
execute(){
o.work()
}
I
work()
I1
I2
I3

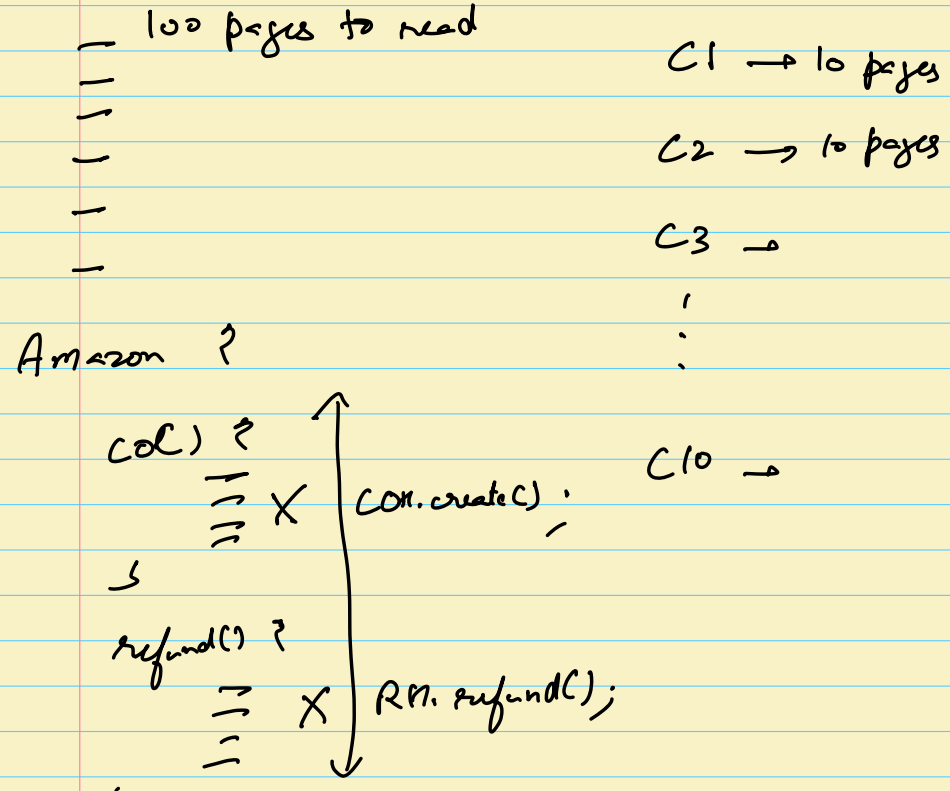
Break = 10:06 - 10:16

Observer Design Pattern





→ Facade was for readability



★ We have to add notification to seller also

- Create another service
- Call the service from Amazon's createOrder

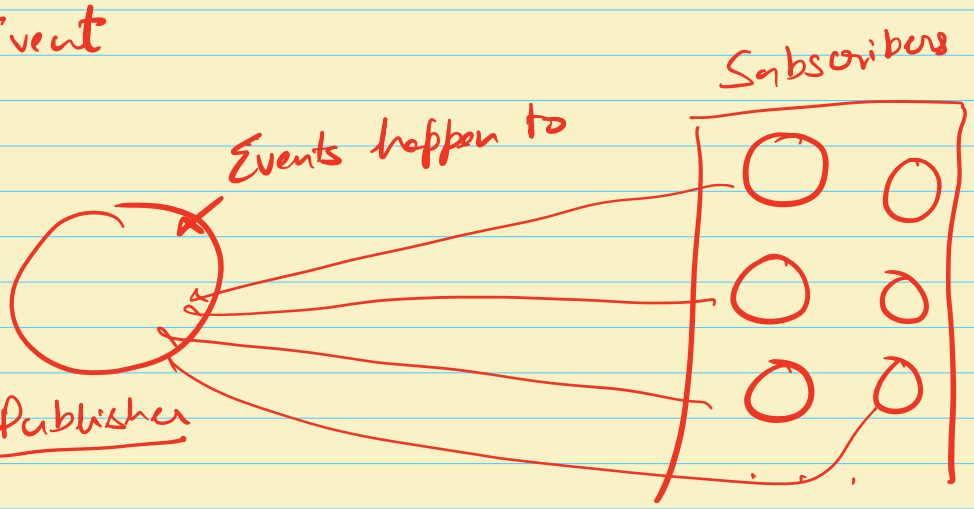
★ This means, to add notification to another service, we will have to write new code, compile it, & redeploy :(

- Current design is not letting us send notifications without coding & deploying again.

Observer Design Pattern allows us to register / unregister new services for notifications about some event at run-time.

Publisher - Subscriber Model

- Publisher
- Subscriber / Observer
- Event



* Subscriber / Observers are allowed to register/unregister for events happening to publisher.

* Publisher sends notification to ~~anyone~~ any subscriber / observer who has registered for the event.

Amazon

```
List<CreateOrderObserver> list;

void regObserver(COObs obs){
    list.add(obs);
}

void unregObserver(COObs obs){
    list.remove(obs);
}

void createOrder(){
    // Code to create Order
    → update inventar
    → sellers
    → logistics
    → send Email
    → send SMS
    → send Push Notif
}
```

interface CreateOrderObserver {

```
void orderCreated();
}
```

EmailService

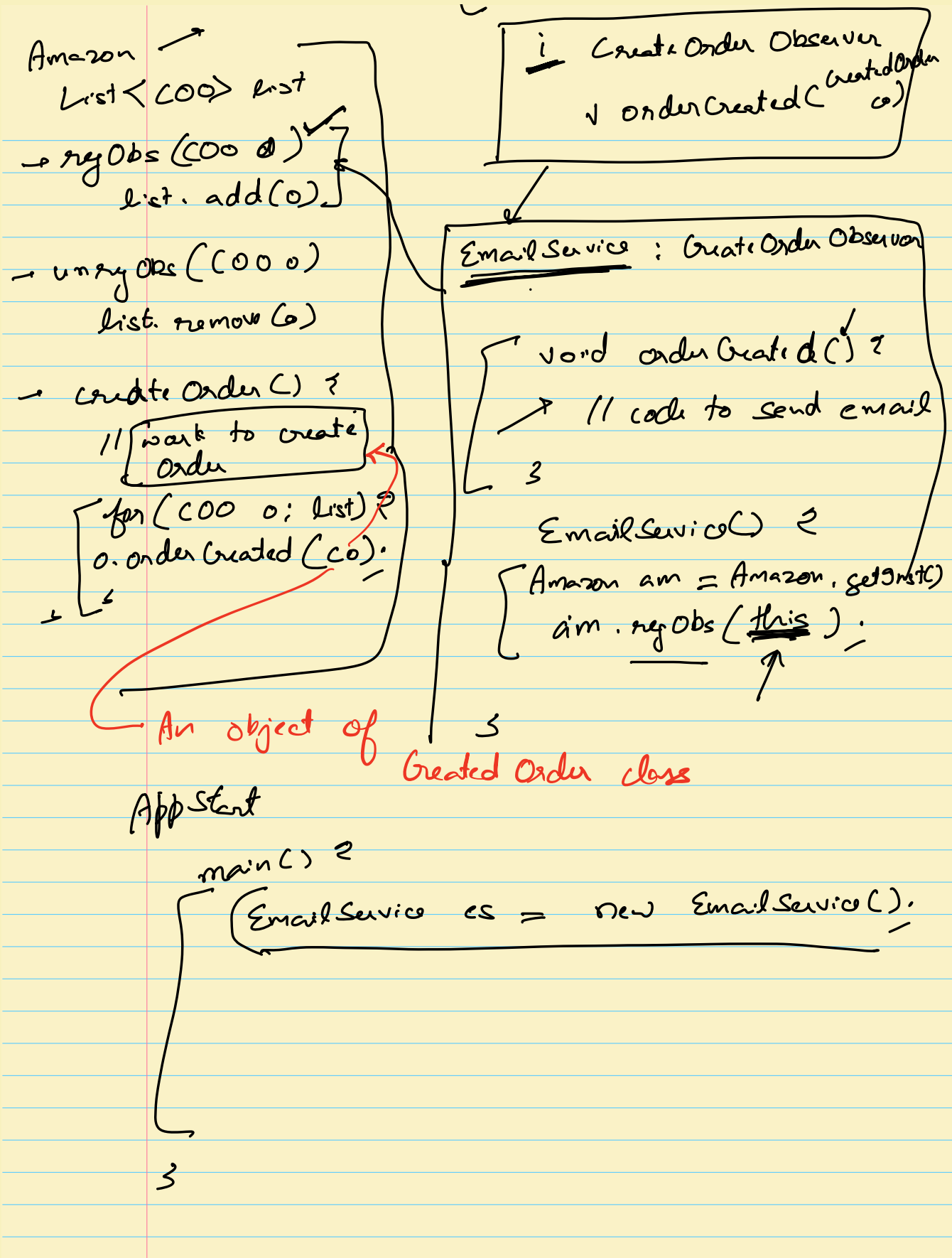
```
orderCreated(){
    // receive notifications
}
```

InventoryService

SMS Service

```
for(COobs o : list){
    o.orderCreated();
}
```

```
class CreatedOrder {
    // prd
    // qty
    // cost
    // address
}
```


Amazon
List<COO> list
regObs(COO o)
list.add(o);
unregObs(COO o)
list.remove(o)
createOrder() {
// work to create Order
for(COO o: List)
o.orderCreated(co);
}
i CreateOrderObserver
orderCreated(CreatedOrder co)
EmailService : CreateOrderObserver
void orderCreated() {
// code to send email
}
EmailService() {
Amazon am = Amazon.getInst()
am.regObs(this);
}
An object of
CreatedOrder class
AppStart
main() {
EmailService es = new EmailService();
}

```
class CreatedOrder {
    // order details
    // qty, price, product
    // address
}
```

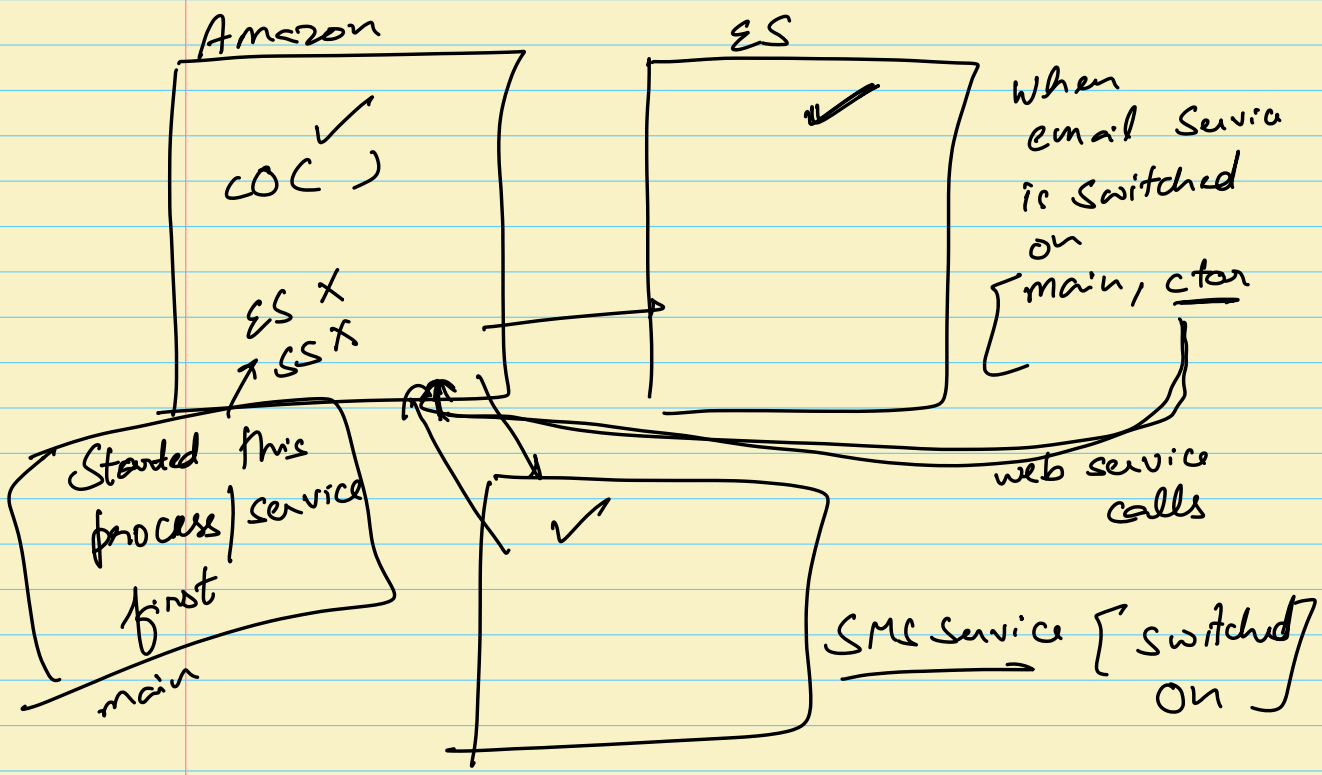
```
i CO Observor

void orderCreated(
        CreatedOrder data);
```

```
Amazon
    List<COO> list;
→ regObs()
→ unregObs()
→ createOrder() {
    // work to create order
    CreatedOrder co
    for(COO obs: list) {
        obs.orderCreated(co);
    }
}
```

```
EmailServic

void orderCreated(
        CreateOrder data) {
    // send emails
    using data
}

EmailSuvic() {
    Amazon am = Amazon.
                getInst();
    am.regObs(this)
}
```

```
Amazon getInstance() {
    // singleton implementation.
}
```

Code of observer



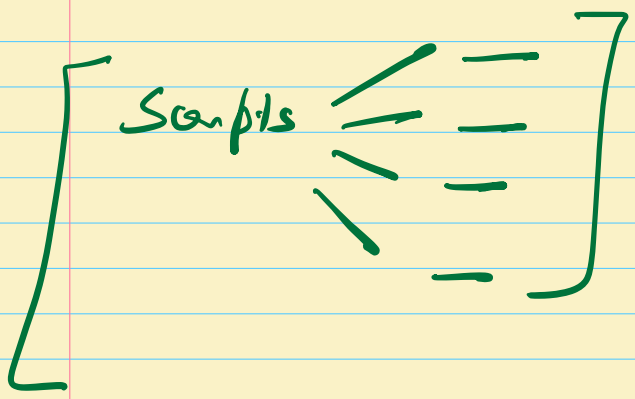
Registering

1. Amazon service turned on first
2. When Email Servic is being switched on, via webservice call to Amazon servic, it will get registered to Amazon's Create Order notifications.
3. When SMS Servic is being switched on ____ same thing as above.

✡ If an order is created now,

SMS service & Email service will be notified.

✡ We need not deploy / redeploy Amazon service to support notifications to new services.



Grunt / Gulp

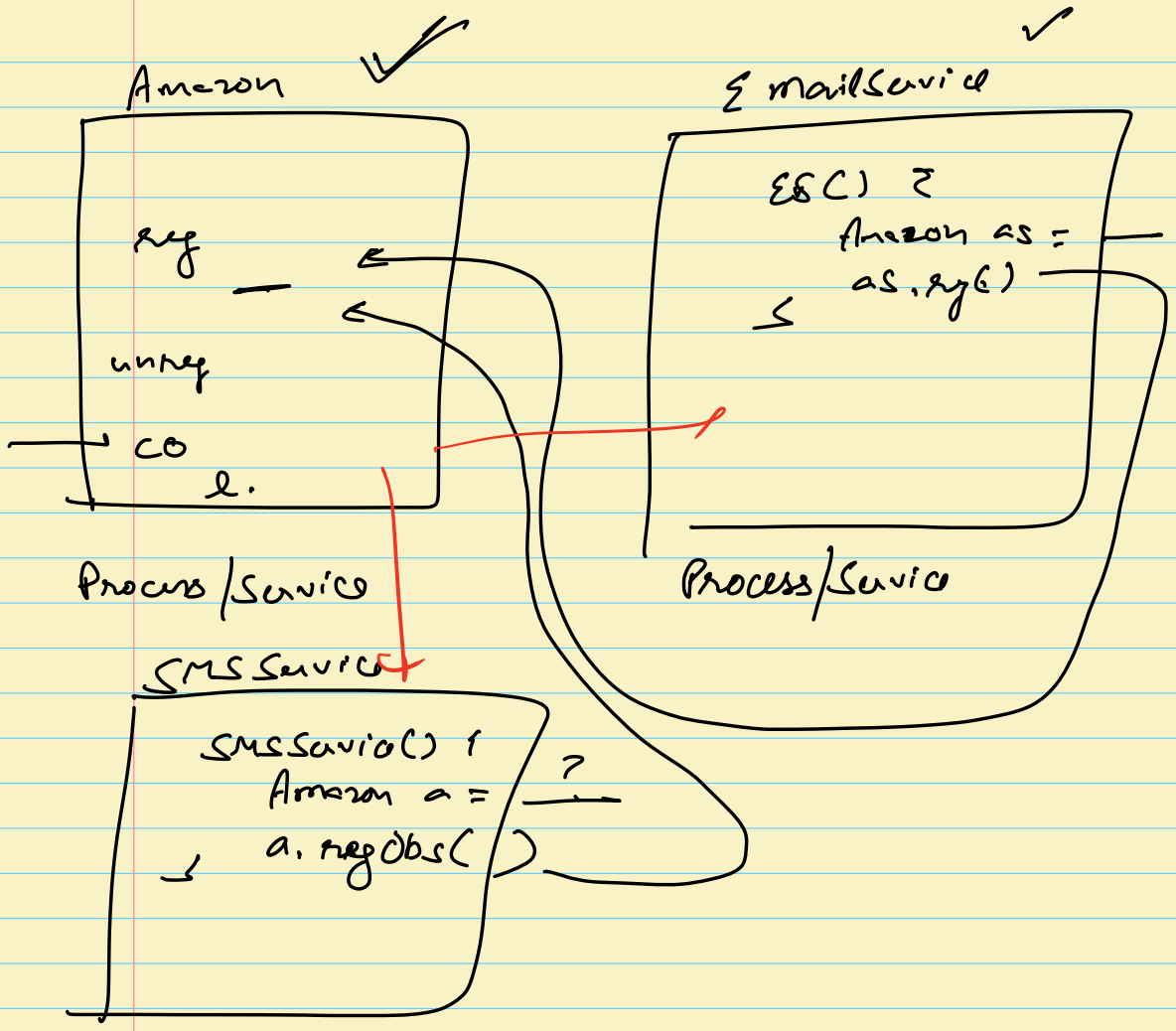
Un registering [HLD Discussion]

→ Heartbeat Mechanism

[A Module to check status of various web services via polling (heart beat check)

If some service is down, this module can]

Unregister


Amazon
reg
unreg
co
l.
Process/Servic
EmailServic
ES()
Amazon as =
as.reg()
Process/Servic
SMSServic
SMSServic()
Amazon a =
a.regObs()