Agenda

1.) What is LLD

2.) Why LLD is important

3.) How to approach any LLD Problem

- → Requirement Gathering
- → Class Diagram
- → Schema Design

low level design of payment apps

4.) Doubts

What is LLD

L → low

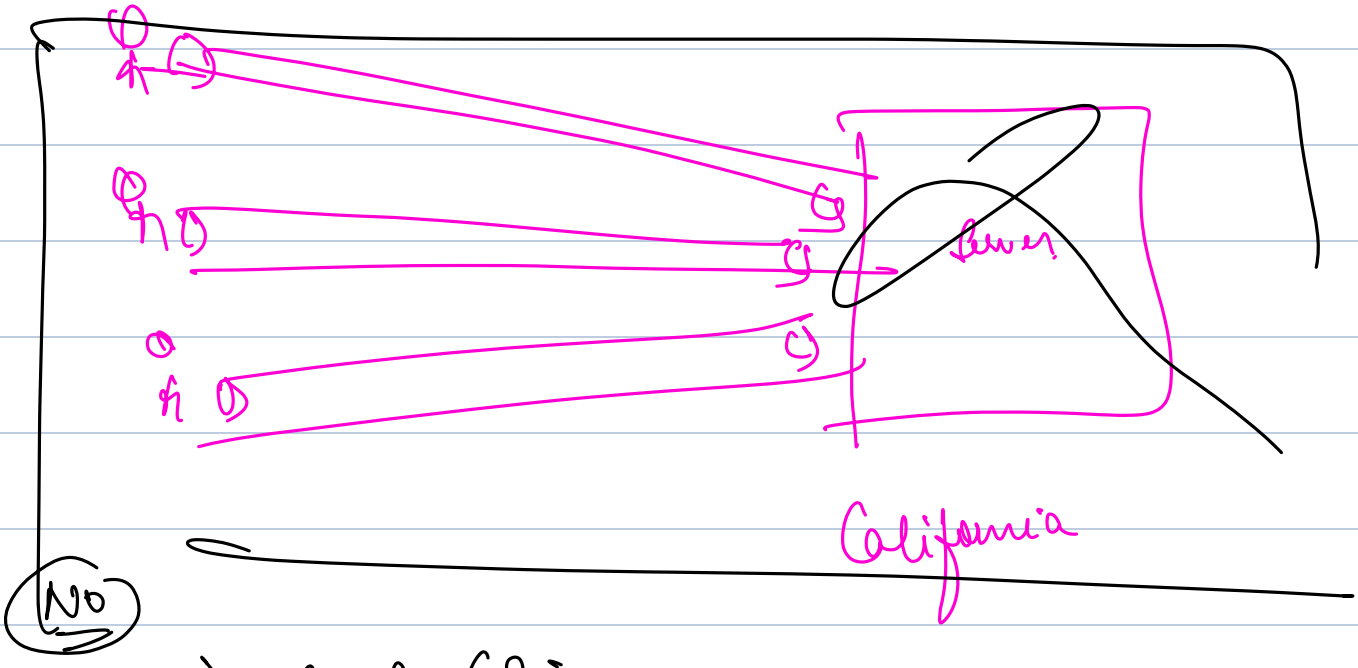
C → level

D → Design.

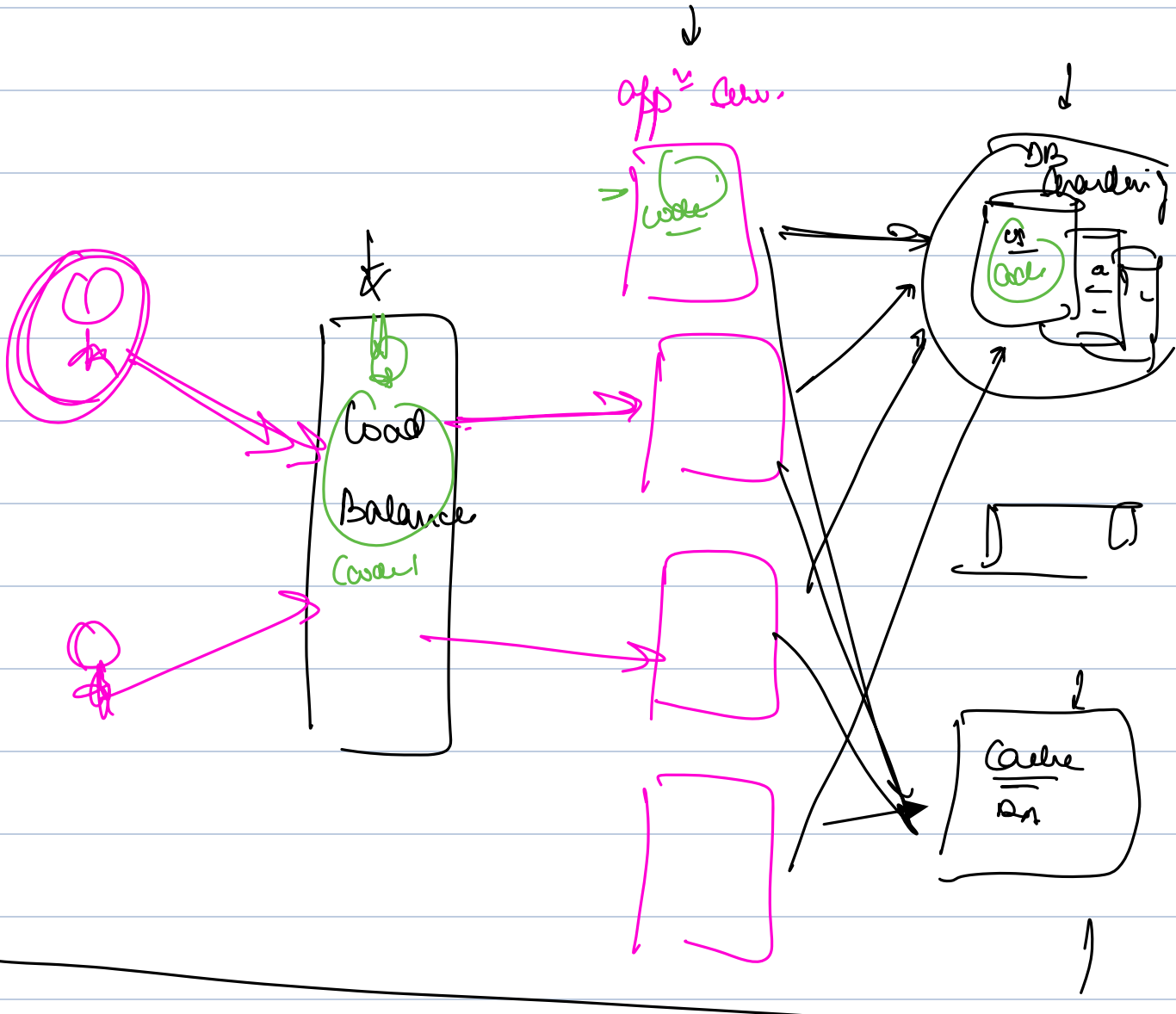
High Level Design → working and interaction of diff infra layers (DB, caches, queue) to serve a SW system at desired scale. at desired efficiency.

SW System

FB Messenger



1.) SPOF (Single Point of Failure)
2.) Bottleneck.
3.) Slow.



Overview → HLD → architecture

Detail → Code

Low level Design

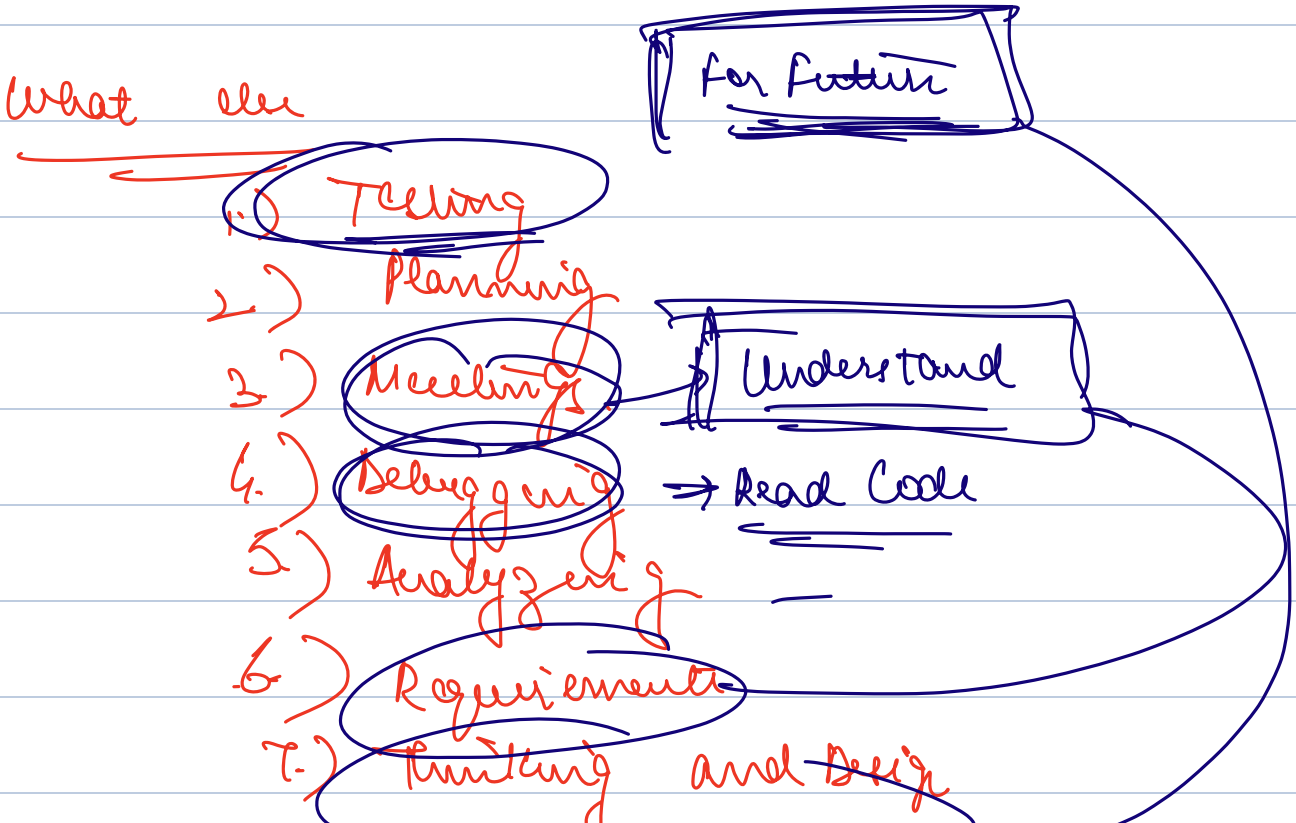
→ how the codebase of a particular SW system is structured

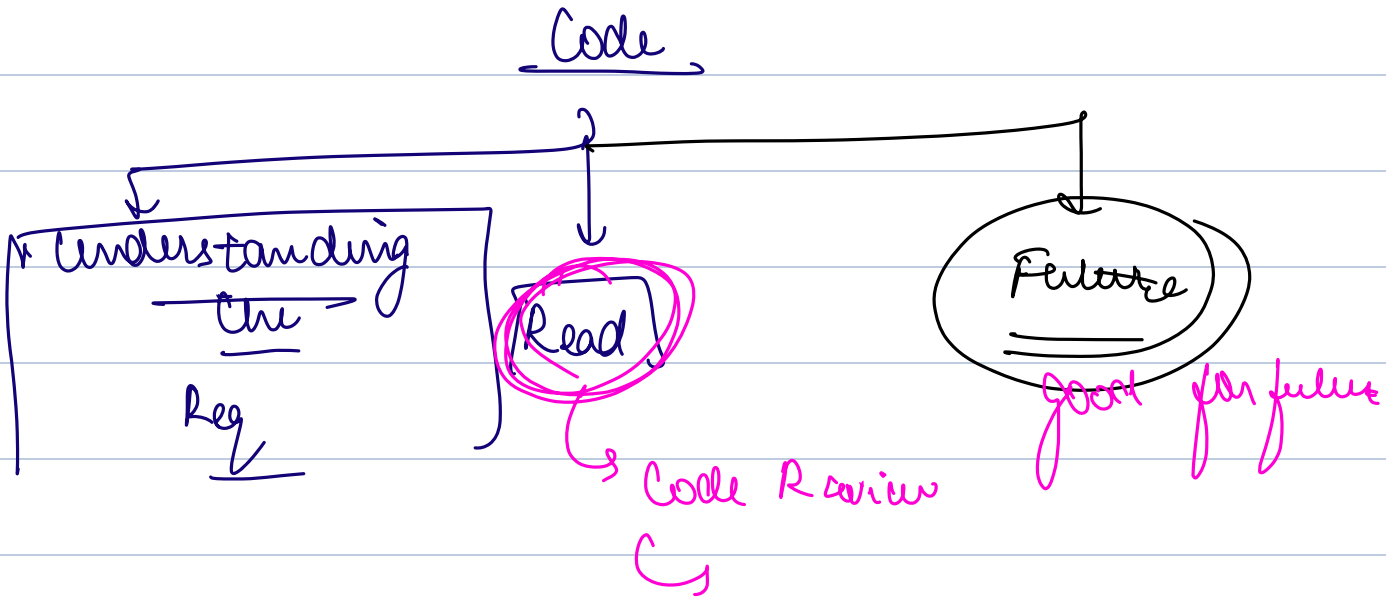
Why ?

S → (1-2 hrs)

{ An avg SWE spends only 1/3rd of their time actually writing code

What else





Low level Design : a methodology of writing good code :

1.) Extensible : easy to add new features to sw system in future.

2.) Maintainable :

1.) dependency security issue

2.) Bug

3.) Platform enhancem

3) Readable (Understandable)

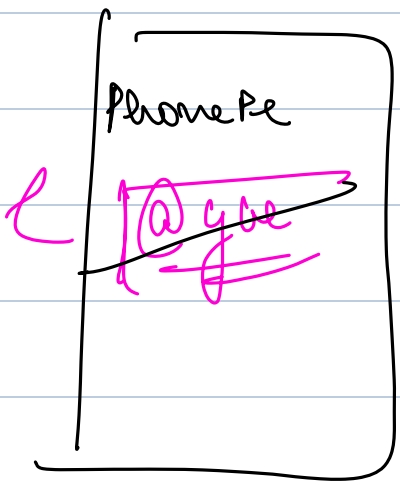
4) Modular

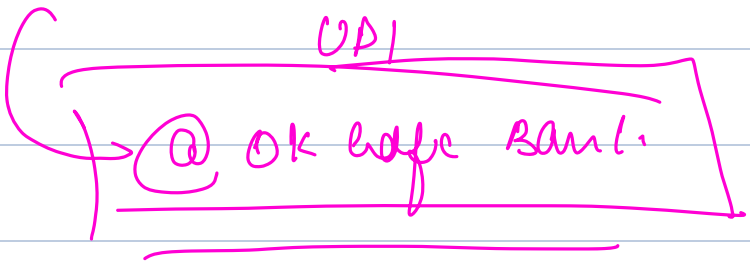
↳ Design Principles (SOLID)

↳ Design Patterns

5 values of a good code design

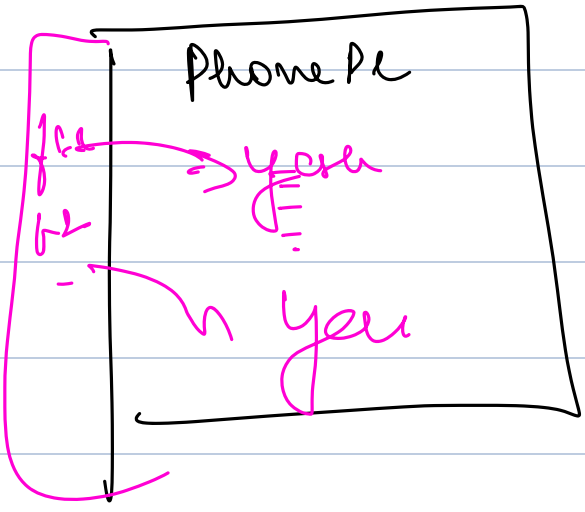
Few years ago, RBI Banned KYBank from doing any online transaction

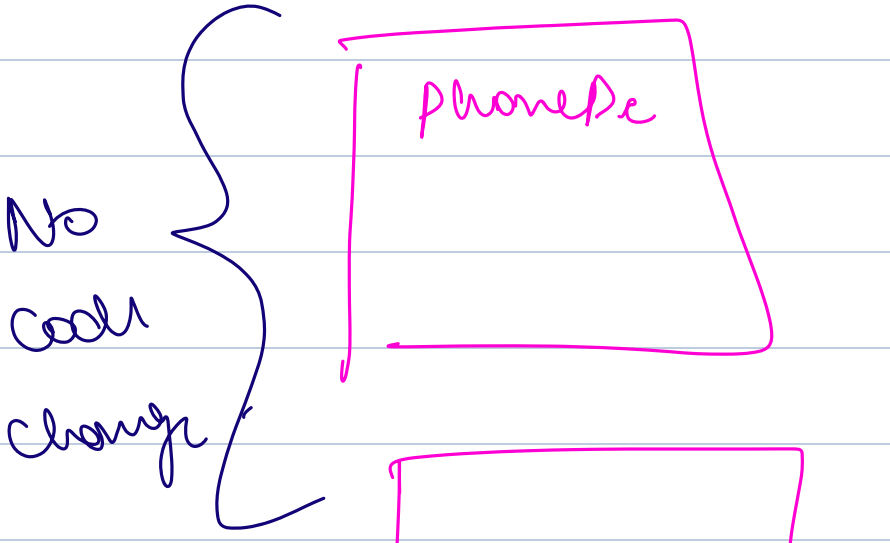


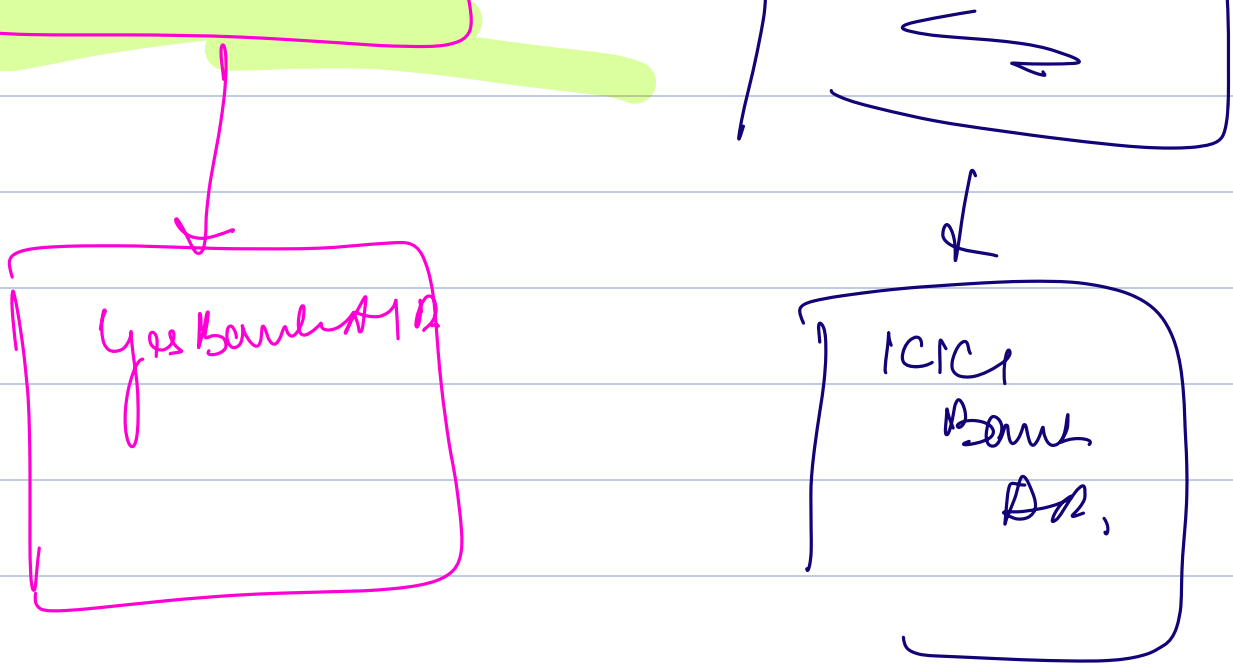


↳ Bank → ICICI

Adapter Design Pattern



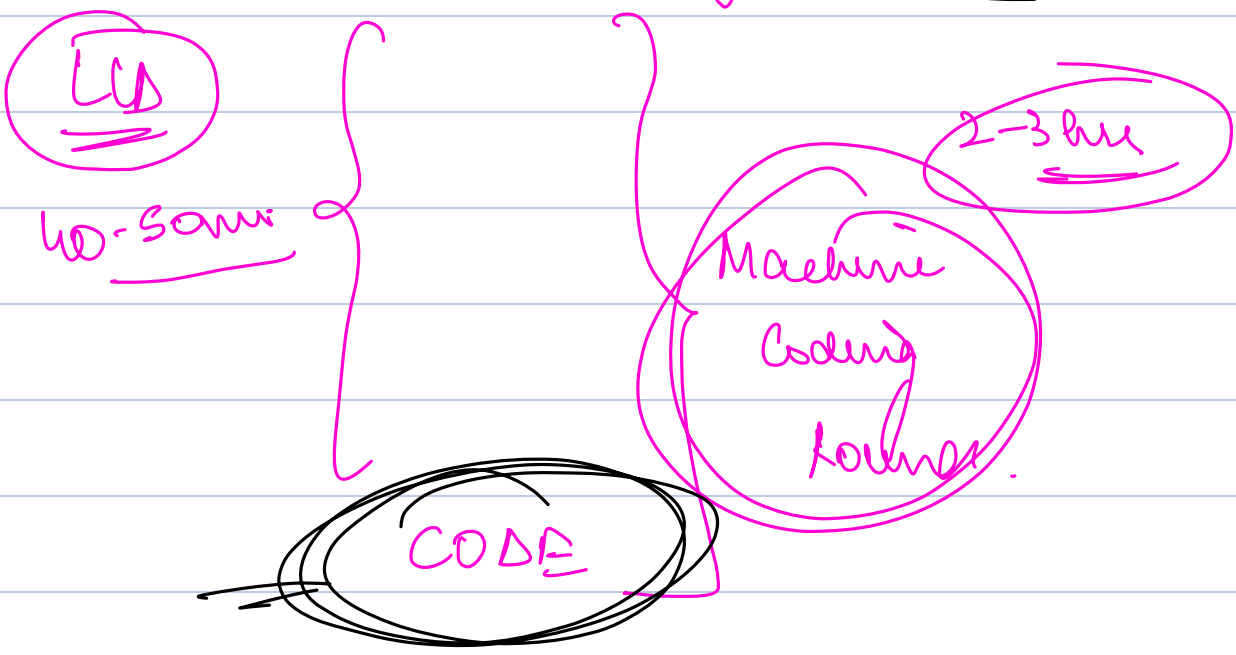




How to approach any LLD Problem

Top Startup SDE I/II —

R1 → LLD / Machine Coding Round



Startups → Ola / Myntra / Cure.fit / Grad / —

How to approach any LLD problem

→ Low Level Design of Book My Show
Splitwise
Google Calendar

1.) Requirement Gathering (dry features to be supported)

Core 4-5 features.

2.) Clarify Requirements

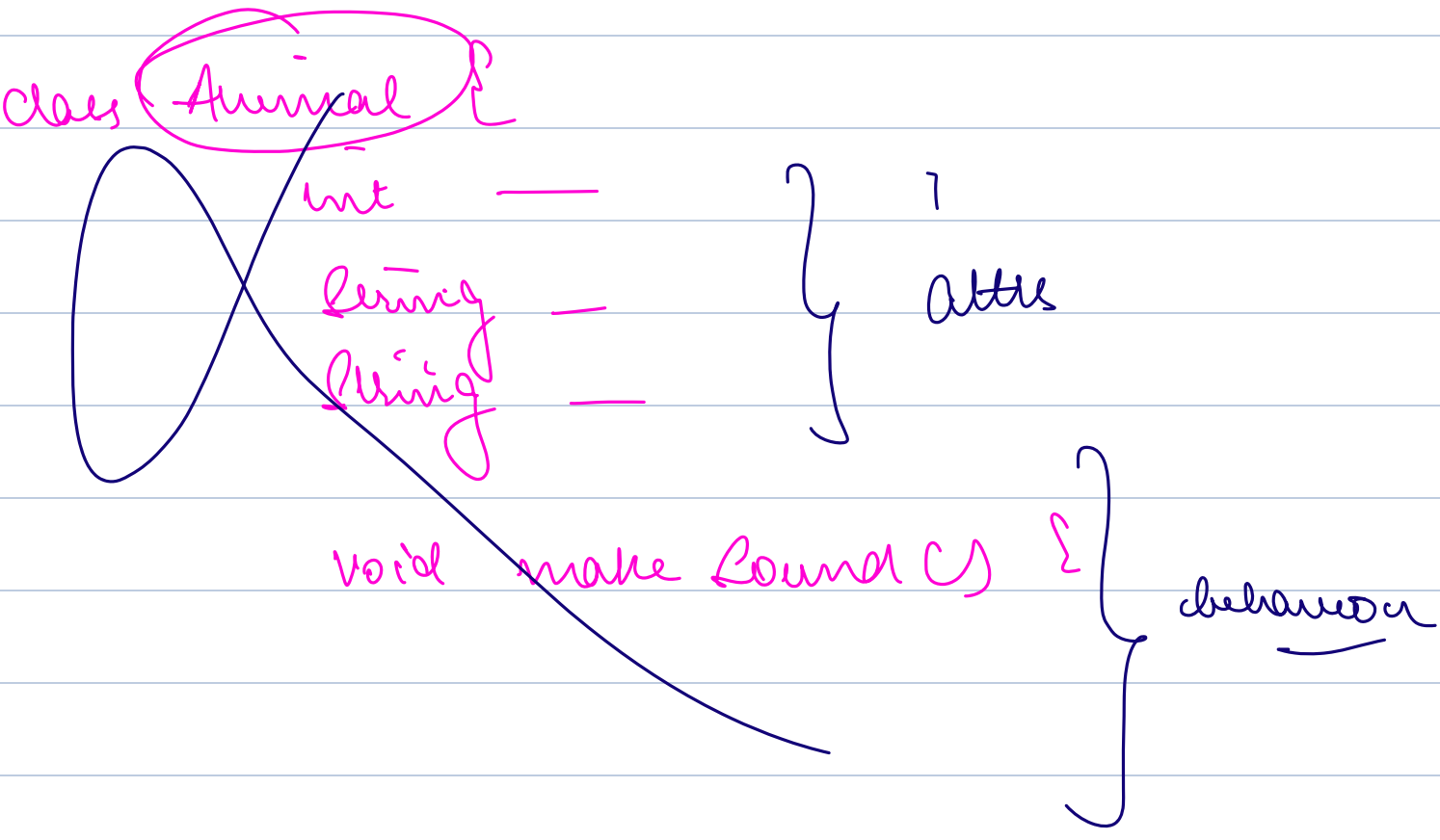
⇒ flow of every feature ASS
should be absolutely U
clear in your mind. MF

3.) Class Diagram

→ dont mention all classes in your system

→ only need to mention ONLY "Models" ⇐

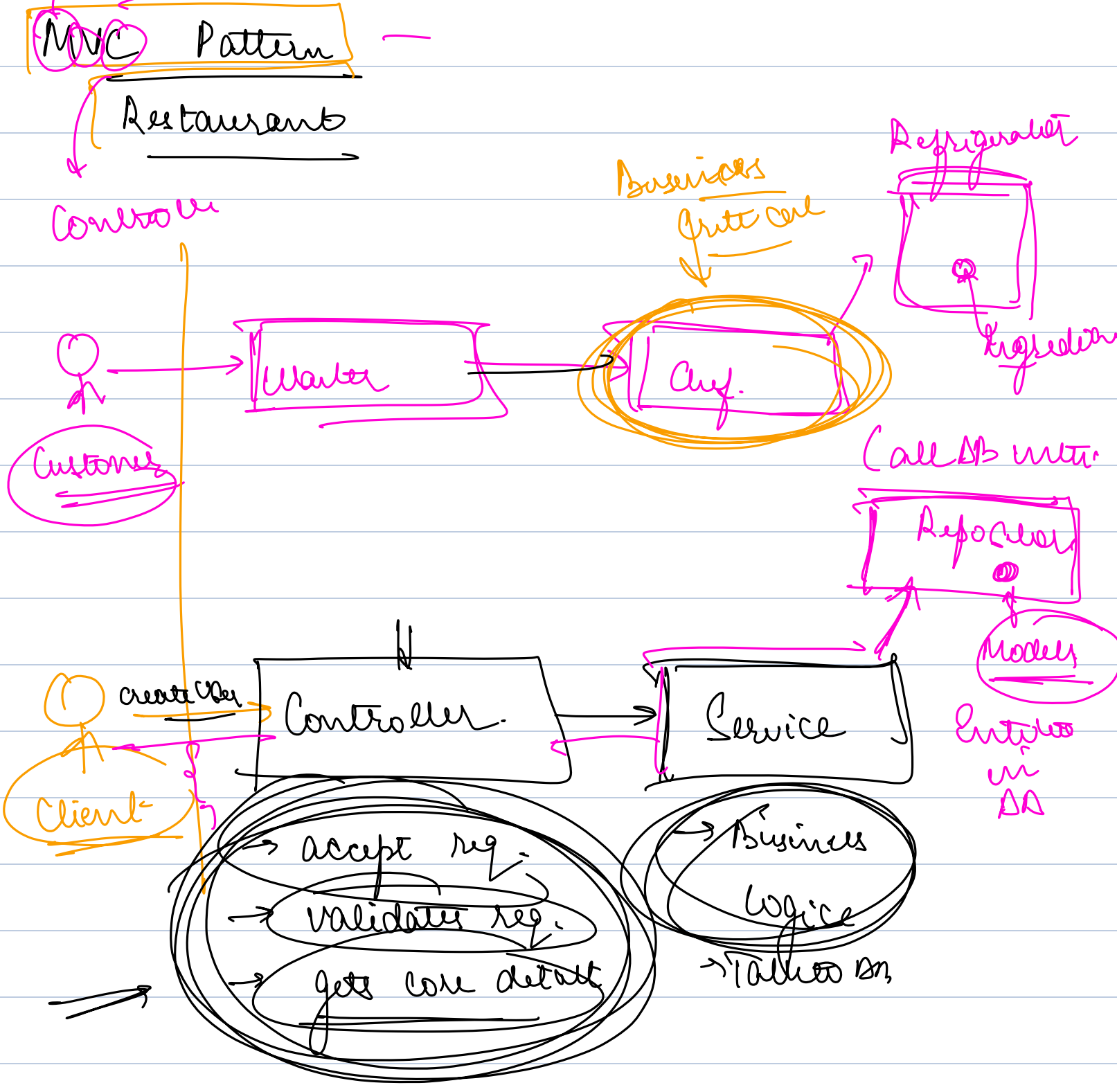
and any classes related to usage of "Design Patterns and Design Principles"



Model → Views (deprecated)

MVC Pattern

Restaurant



DB Schema Design || (Assume relational DB)

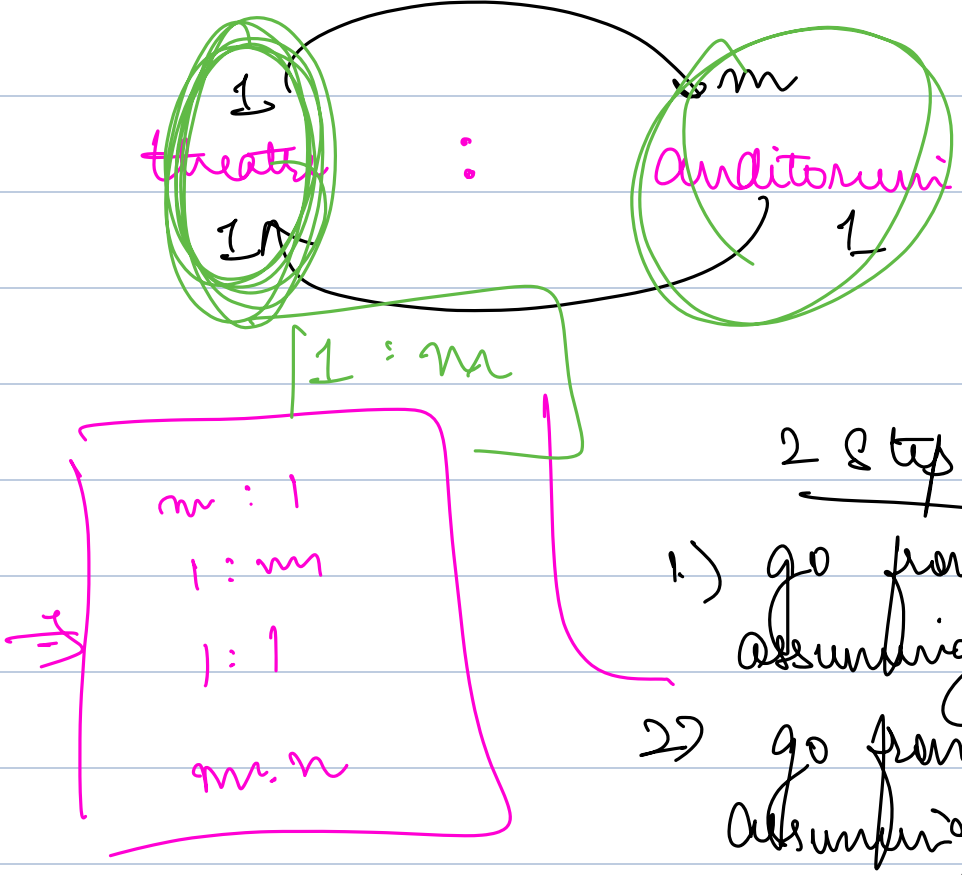
→ tables

→ attributes

→ relationships b/w tables
& cardinality of relation

- 1 : 1
- 1 : m
- m : 1
- m : m

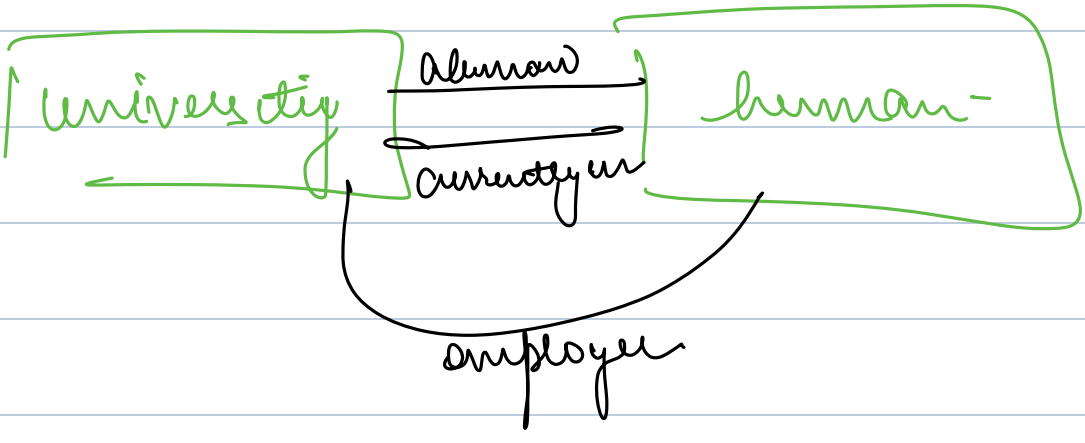
→ any table for rel^n

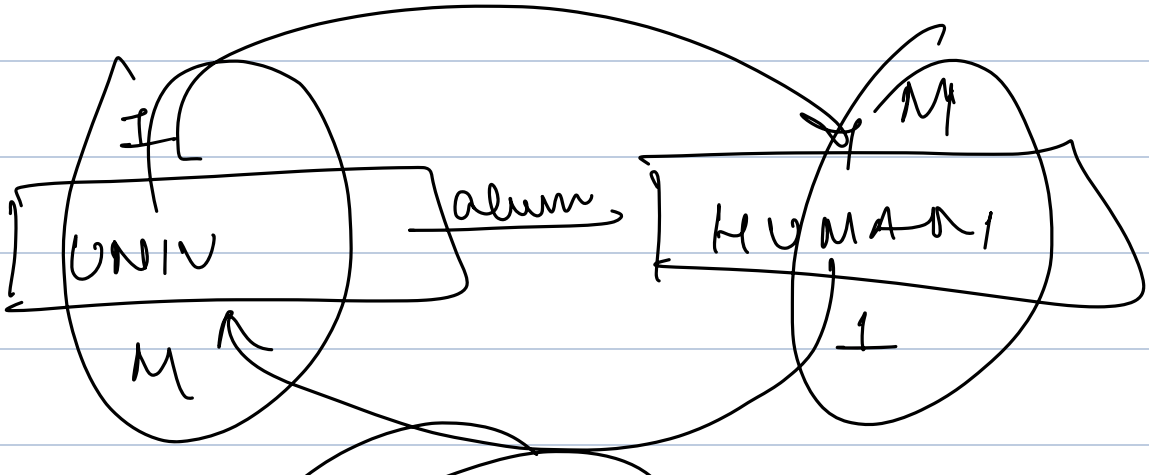


2 steps approach :

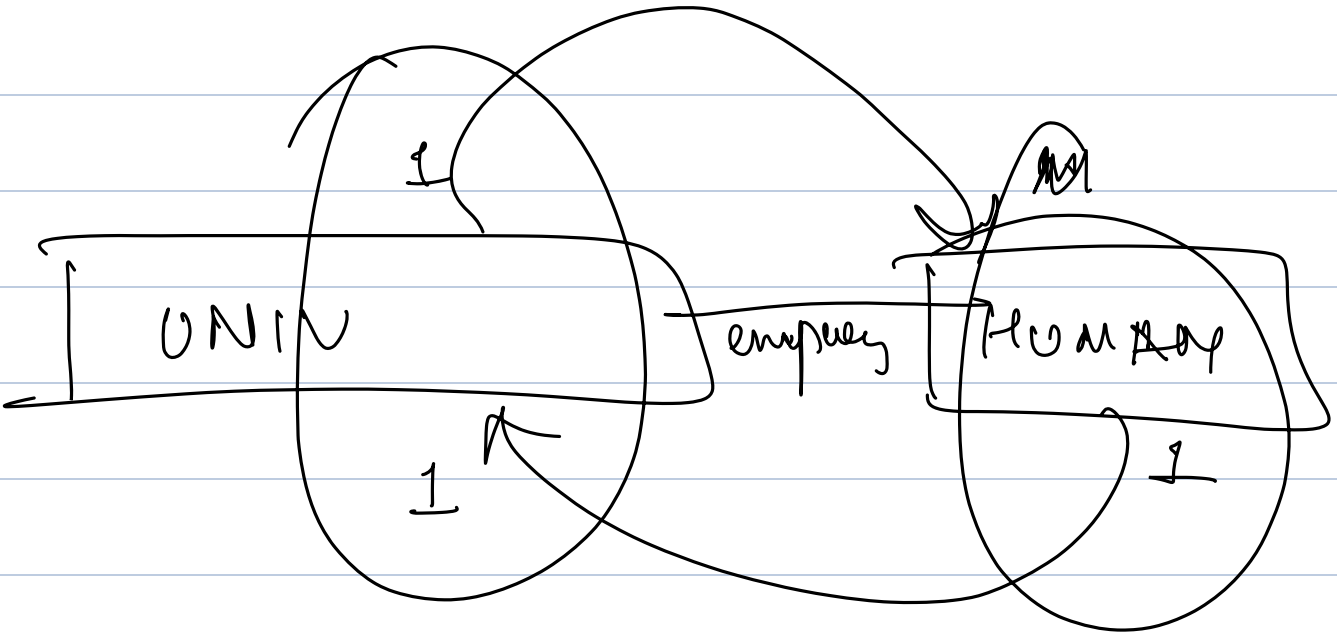
1.) go from L to R assuming 1 on L side

2) go from R to L assuming 1 on R side





M : M



1 : M

Ex of Payment Apps

1. GPay, Phonepe
2. Razorpay, Pay U
3. Wallet Kyc

0- Overview → clear what kind of system are we designing

1.) Gathering Req ⇒ 6-8 Min

2.) Clarifying Req

→ ask q^n along with suggⁿ

→ only ask q^n that will have implication on design

'Wien and Martin'

3.) Class Diagram

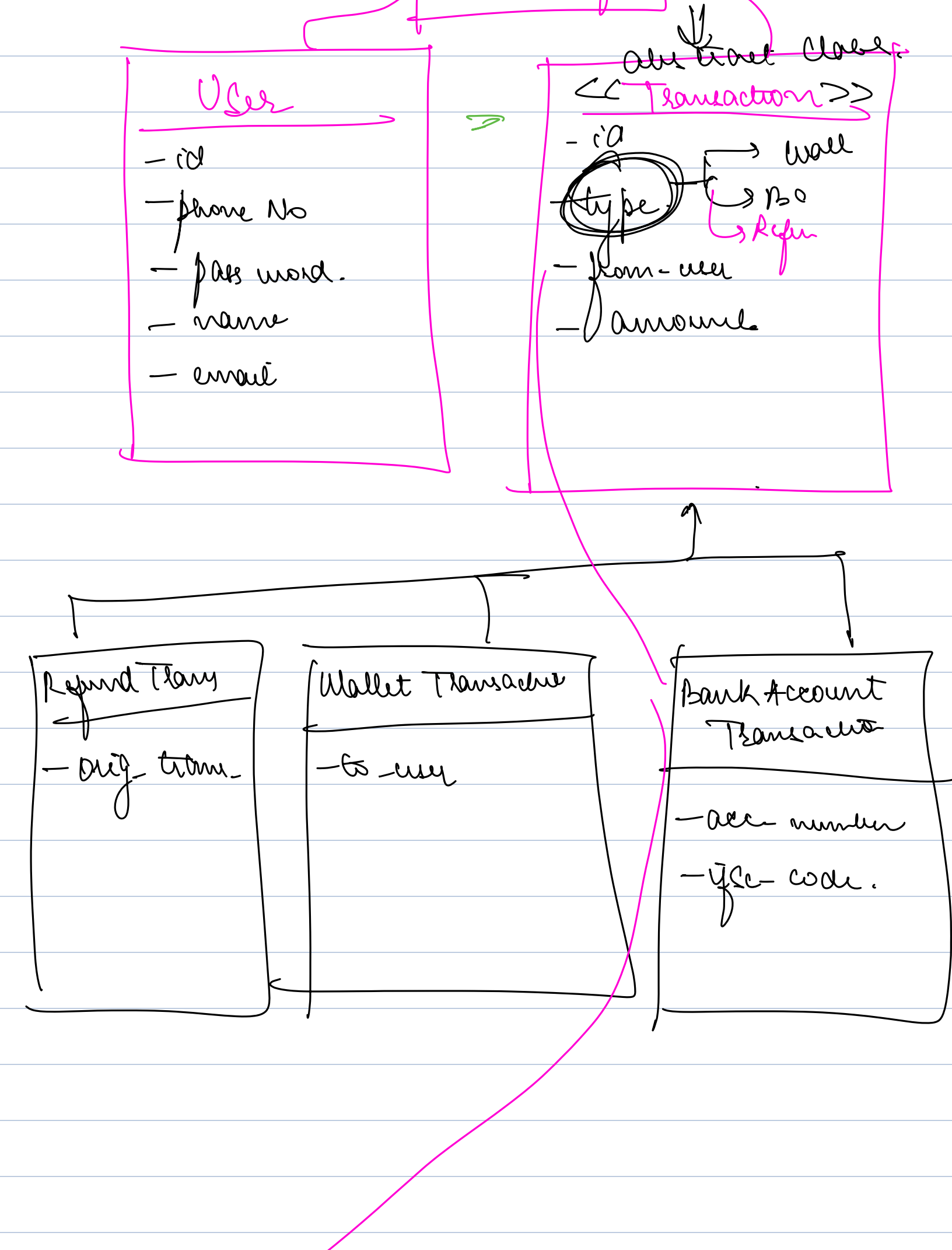
for every req:

if req has "Noun" (an entity about which you will be storing details in DB):

create a class:

BaseModel

- id
- createdAt
- lastModifiedAt


User
- id
- phone No
- password
- name
- email
Abstract Class
<<Transaction>>
- id
type
Wall
BO
Refun
- from-user
- amount
Refund Trans
- orig_trans_
Wallet Transactio
- to_user
Bank Account Transactio
- acc number
- ifsc code.

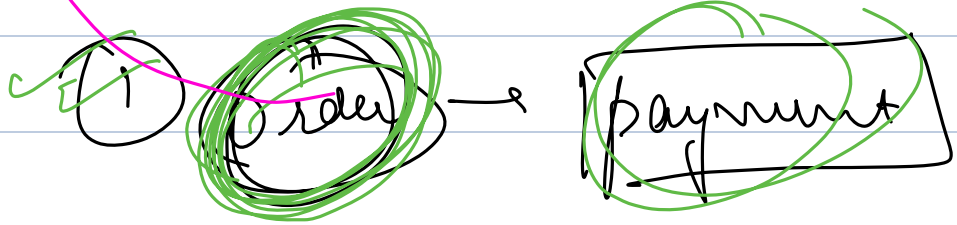
Switch (transaction) {

Case WALLET:

Case

}

E-COM



Yes

2. Payment → Order. No

Eg Partial Payment

Order ₹200

↳ discount — coupon ₹20

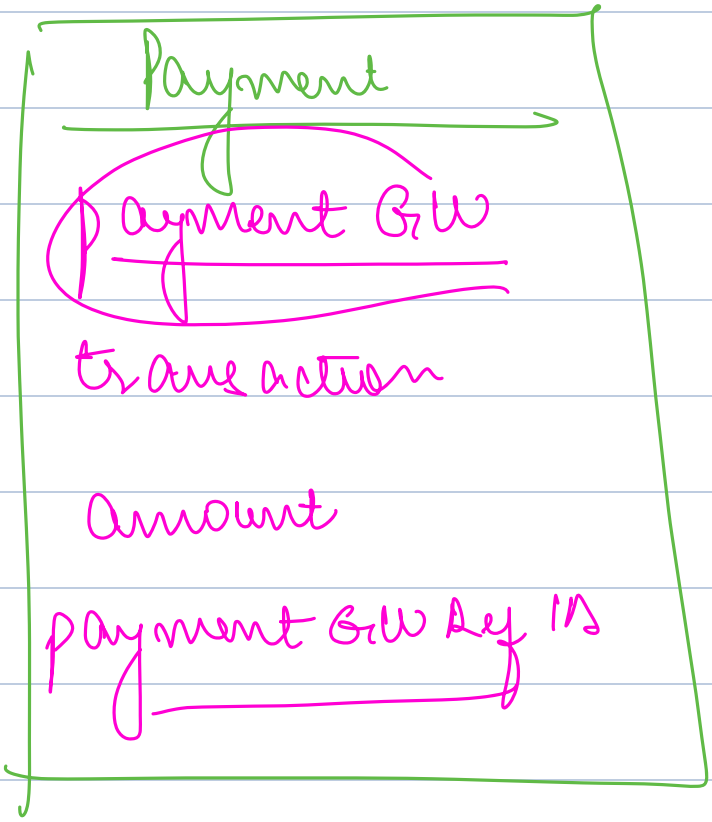
↳ credit_card ₹180

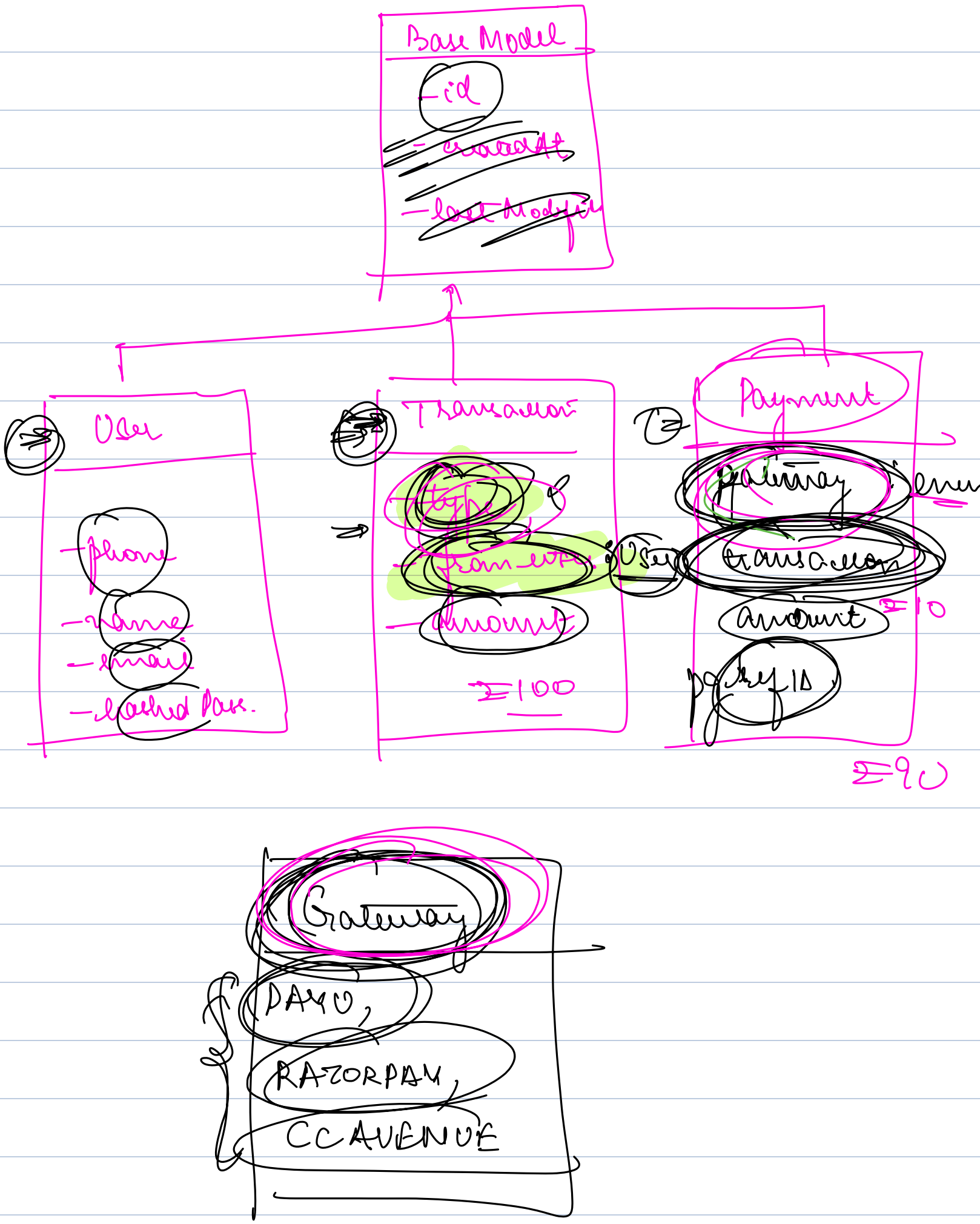
1) Order

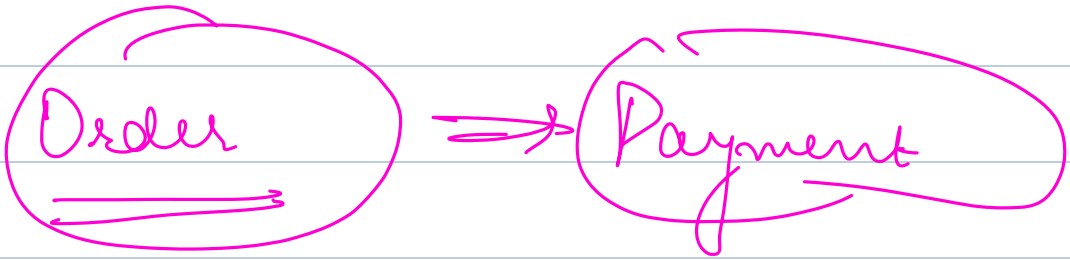
2) Pay for Order

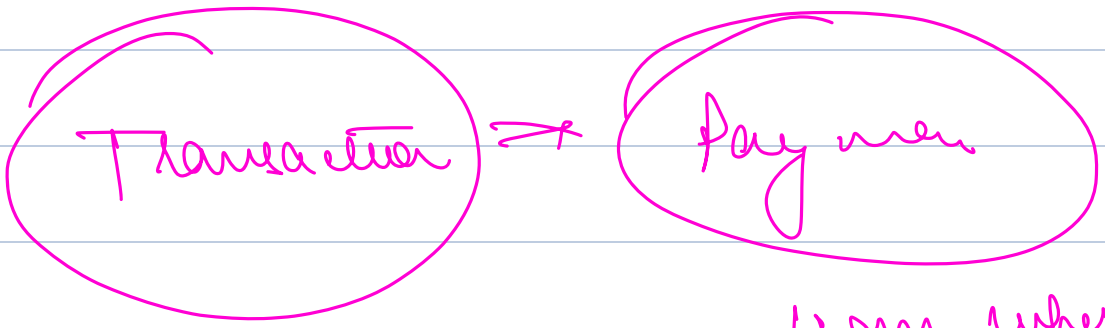
1) Create Transaction

2) Pay




Base Model
id
User
phone
name
email
Transaction
Payment
Gateway
transaction
amount
Gateway
PAYU,
RAZORPAY,
CCAVENUE





Send this much
to them

from where.

DB Schema Design

for every class in CD:
    create a table
    for every primitive attr (non-class attr):
        create them as attr of corresponding table

for every class:
    for every non primitive attr:
        find cardinality of rel^n
        new table / rep as per cardinality

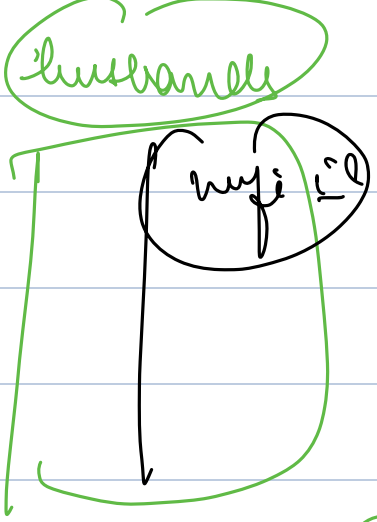
every enum will be rep via a rep table with only 2 attr:
id | value

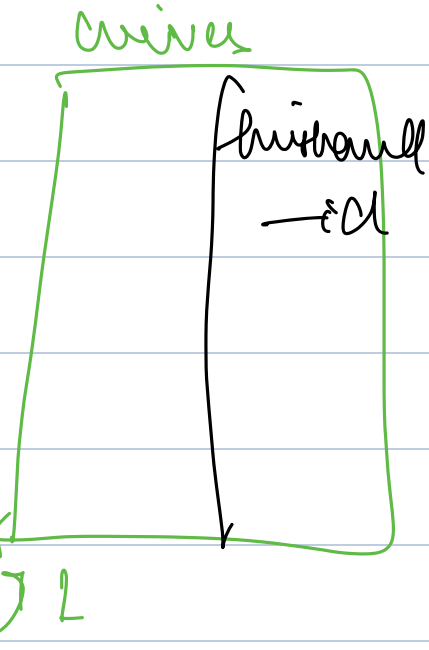
1 : 1

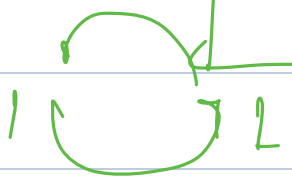
1 : m

m : 1

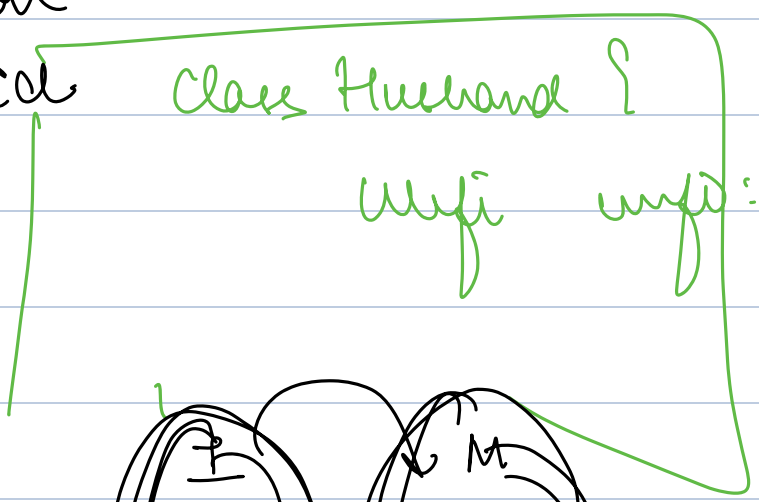
m : m

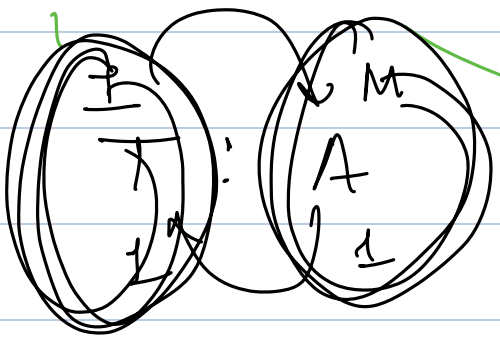




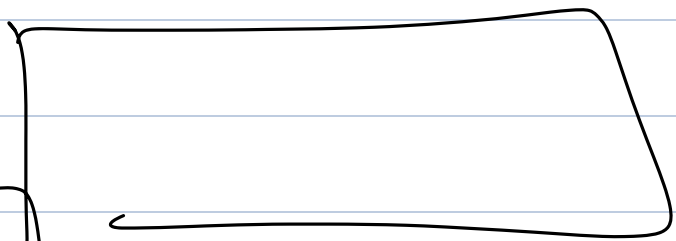


id of any
1 side or
other side





Theatre



1 : m

or

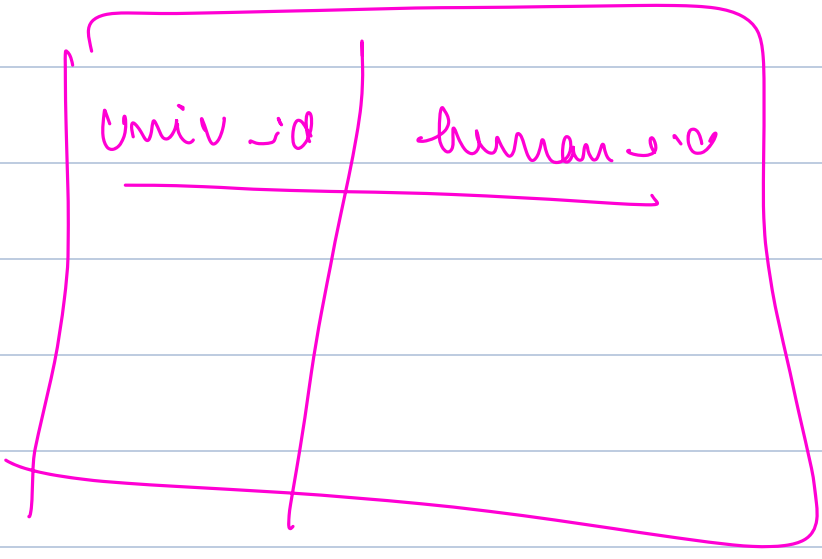
m : 1

Put id of
'1' side
on 'm' side

Auditorium

m : m = Separate Mapping Table

Univ - Alumni



Cardinality	How
1 : 1	id of any 1 side on other side
1 : n	id of 1 side on n side
n : 1	
n : n	mapping table

Users

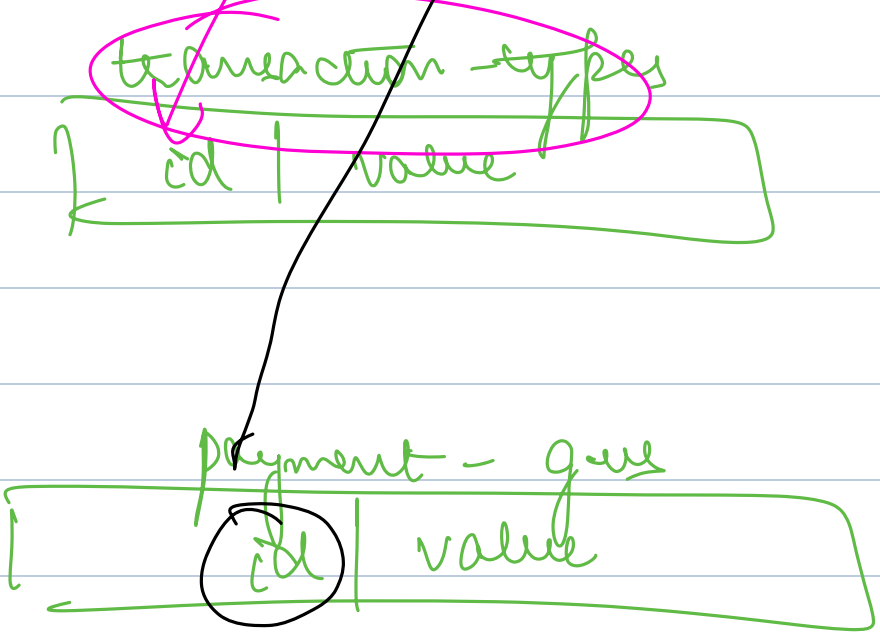
id	name	email	pwd	last login

transactions

id	amount	from_user_id	trans type - id

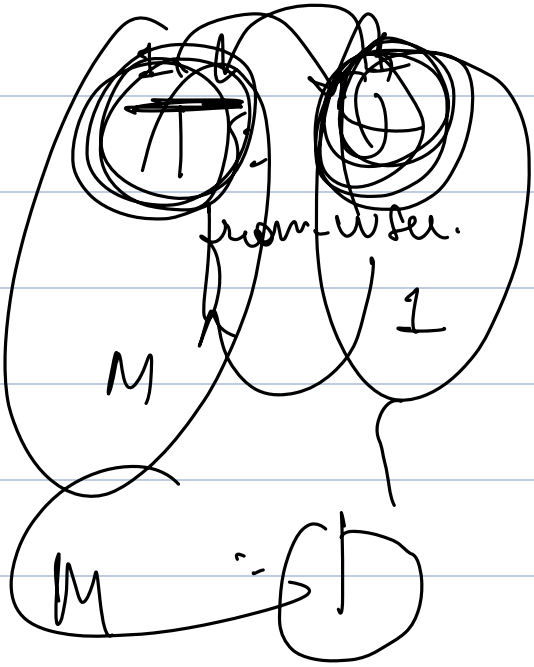
payments

id	amount	ref id	pay-gw - id	trans - id



add to learning

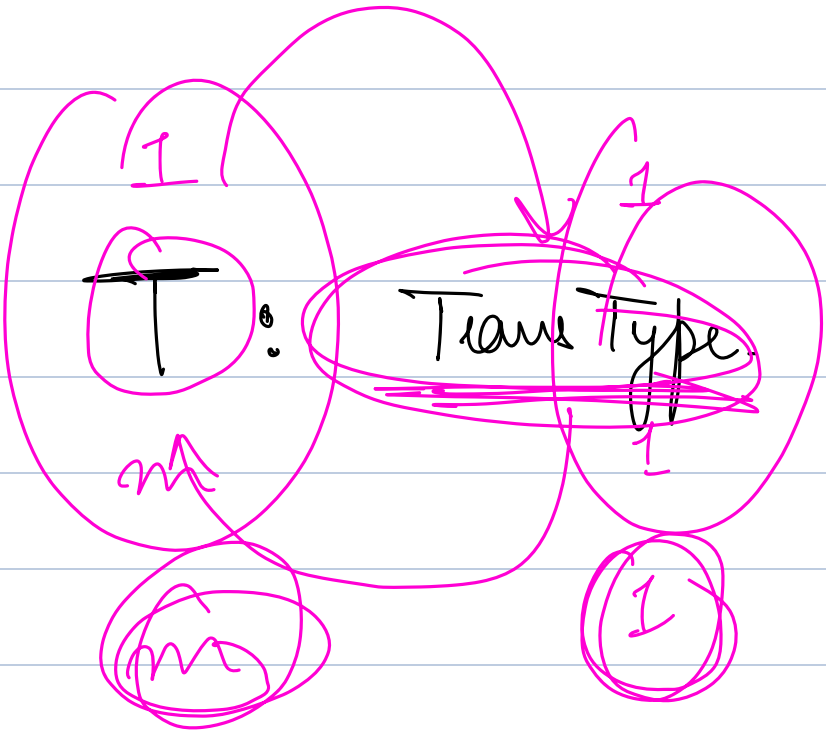
- SOLID Design Principles
- Design Patterns
- Machine Coding. Case Studies

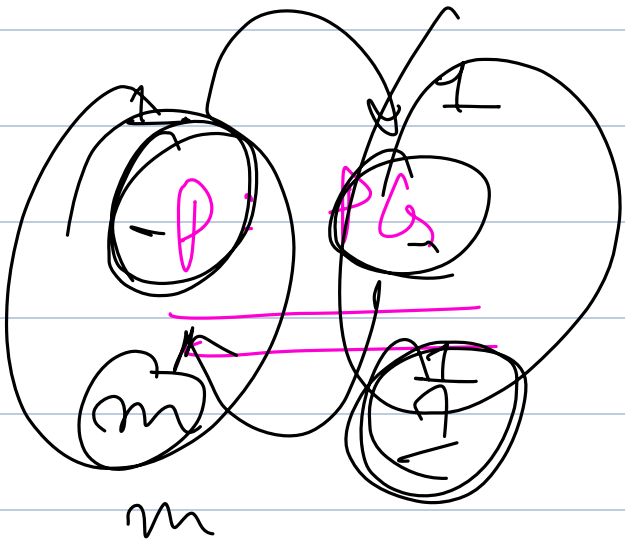


1 : 1

1 : m

m : 1

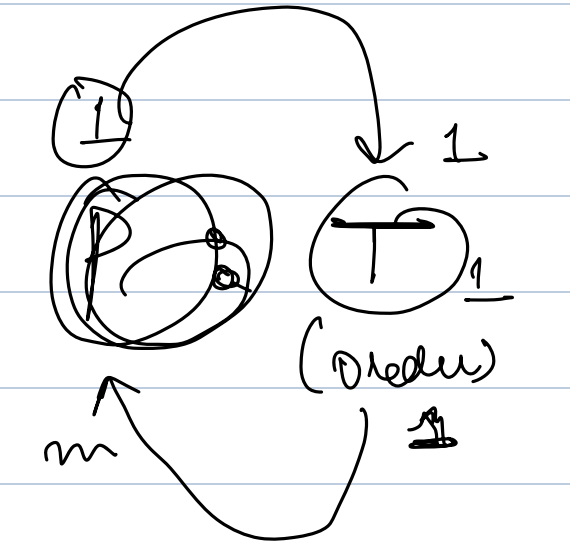




10 =>

30 =>

70 =



m :