Agenda :-

i) How important is LLD ?

ii) LLD in interviews

iii) How start with a design problem

iv) Design steps

v) Design LLD for Payment Apps

- Class diagram
- Schema Design
- Tips to code

⇒ Different types

3 types

Tech / framework Theoretical	Design Round	Machine Coding Round
→ TCS / Infosys / Wipro / PwC / Deloitte kind . — — —.	→ Microsoft, Amazon, Atlassian, Meta, Myntra,	→ Swiggy, Flipkart, CRED, Scaler . .
→ theory based / syntax based	→ Single line problem Statement	→ Problem Statement end to end
→ minimal to no coding	→ Requirement gathering	→ Design it
→ minimal to no design	→ Design the system end to end	→ write end to end functional code
→ 45 mins	→ Class Diagram / Schema Design / Code Structure	→ write test
		→ 120 mins

→ Design patterns
→ Future requirements
→ Go wing

[2 - 2.5 hrs]

→ How to start with a design problem:-

→ LLD => Low level Design

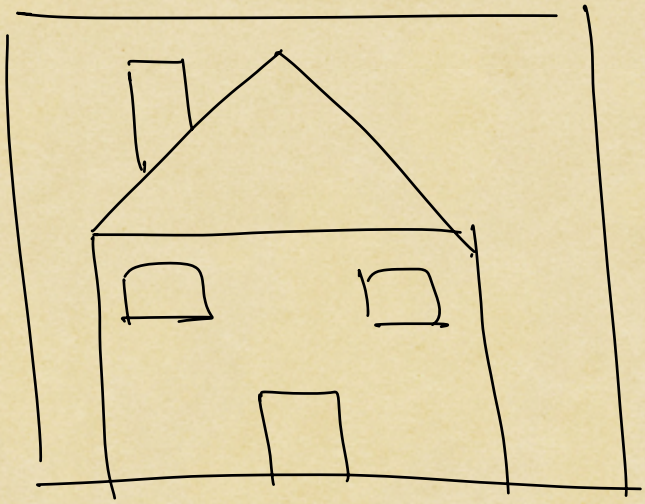
↓

tells us how to write the code

↓

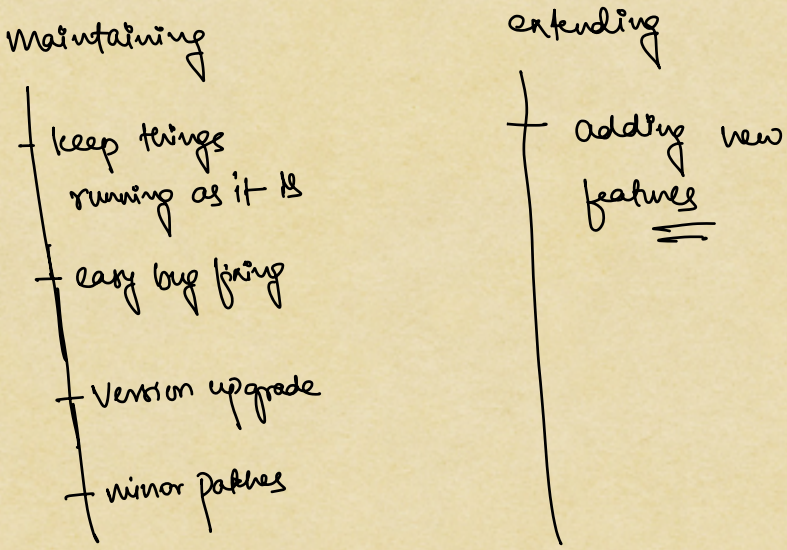
how to structure our code so that it is

- easy to read
- easy to maintain
- easy to extend.



↓

pillars, beams, walls, thickness, weight of ceiling.



→ Requirement gathering

→ Start the design

- use case diagram
- class diagram
- schema design
- structure my code

STEPS TO CREATE LLD FOR ANY PROJECT

STEP-0 ⇒ Get an overview:

Align yourself with the thought process of the interviewer/PM.

Overview

You already know about the system

* align your understanding with the expectations of the interviewer

You don't know anything about the system

* ask the other person to give an overview of the system

* What do you want me the design?
  - → Entities
  - → Entities & relationship
  - → Appn [end to end]

* Do you want me to persist data?
  - ↓
  - Stored in DB
  - ↓
  - do you also want the Schema Design?

design of DB, about how to store data.

* How will we interact with the system?

→ REST API

→ Command Line Interface

→ Hard Coded

STEP - 1 - Requirement Gathering

gather and classify requirements

→ Suggest ideas with rationale and ask if we should support that

→ 5-8 core features

→ Try to visualize the system

→ Jot down the pointers

→ For every feature, try to think about edge cases and future requirements

⇒ Requirements for payment app:-

USER

* Signup → email → password
or phone → OTP

* Login → email → password
or phone → OTP

* Add profile details and do KYC :-

→ KYC
- name
- phone no.
- PAN
- Aadhaar

→ Bank Details
- Bank A/c no.
- IFSC code
- Branch Name
- A/c holder Name

* Send money to others via

- UPI Id
- phone no.
- Bank A/c
- QR code

Only these 2 are supported for now

MVP → Minimal Viable Product

Wallet

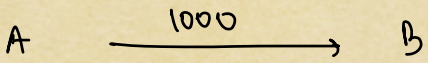
* Source of payment -
→ Debit card
→ Credit card
→ UPI

* see history
  → Pagination
  → 10 transactions/page
  → most recent first

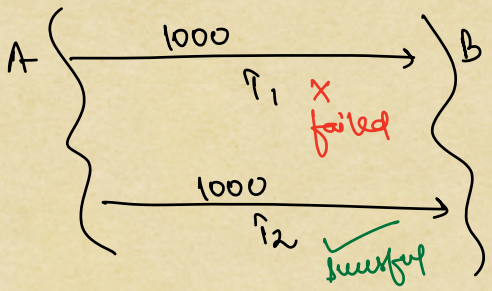
* how do you handle transactions :-

i) We want to allow retries.

ii) We should allow multi-transaction payment

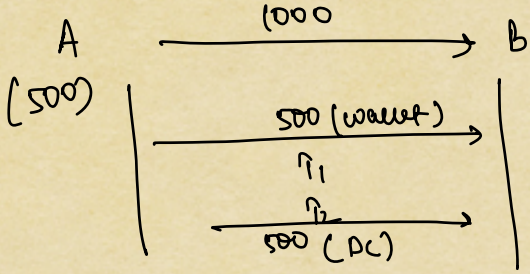


a)



→ 2 transactions to complete 1 payment

b)

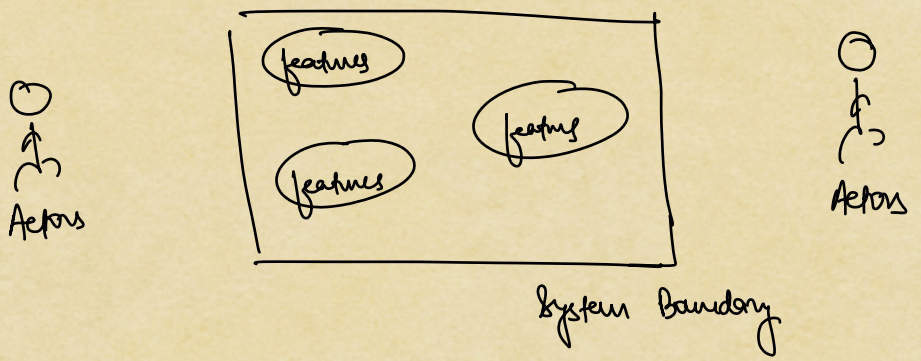


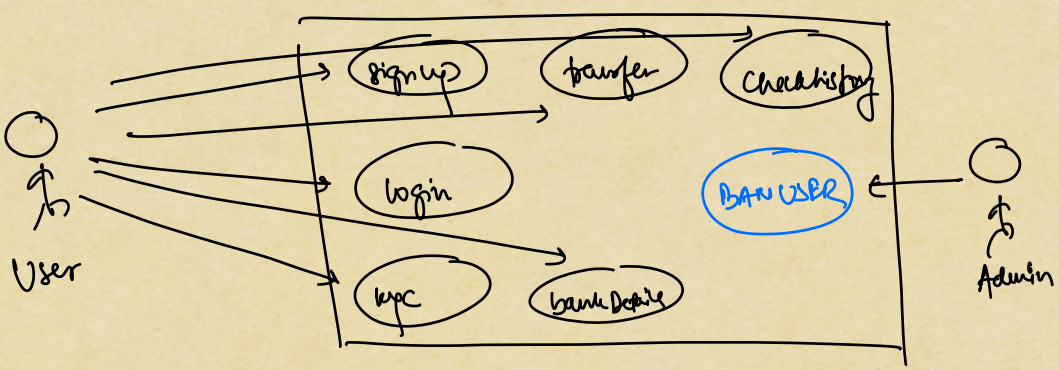
→ 2 transactions to complete 1 payment

Atomicity

iii) If sender/reciever has any problem and any transaction fails, we will cancel the whole thing and rollback.

⇒ STEP-2 USE CASE DIAGRAM





↓

Use case diagram should only contain the requirements discussed earlier.

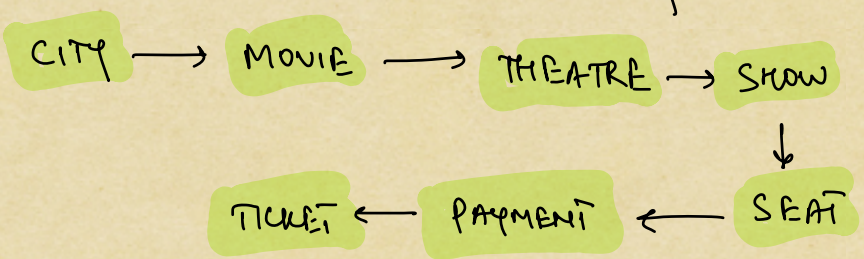
STEP-3 - CLASS DIAGRAM

- entities
- relationships b/w entities
- enums, interfaces etc
- design pattern used

while following SOLID principles

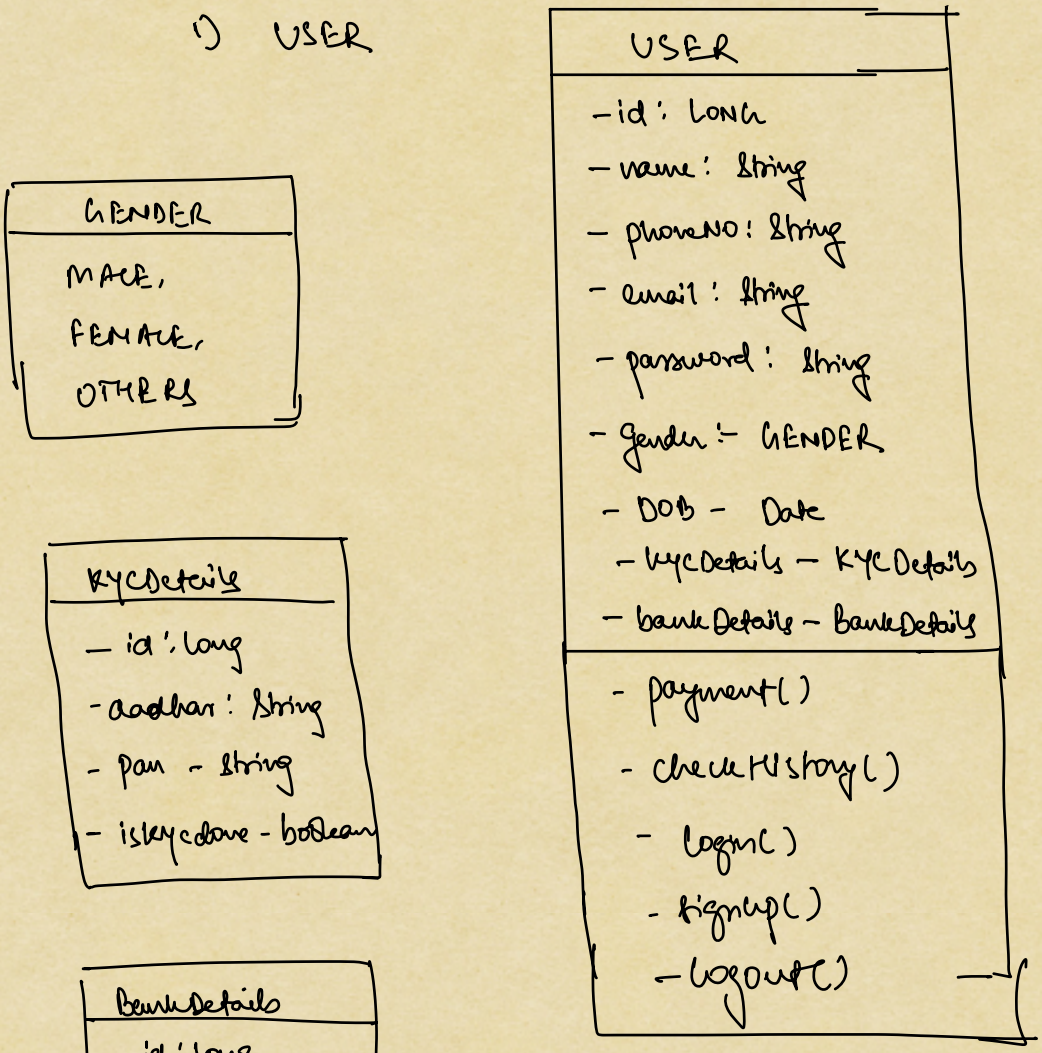
- NOUNS FROM REQUIREMENTS

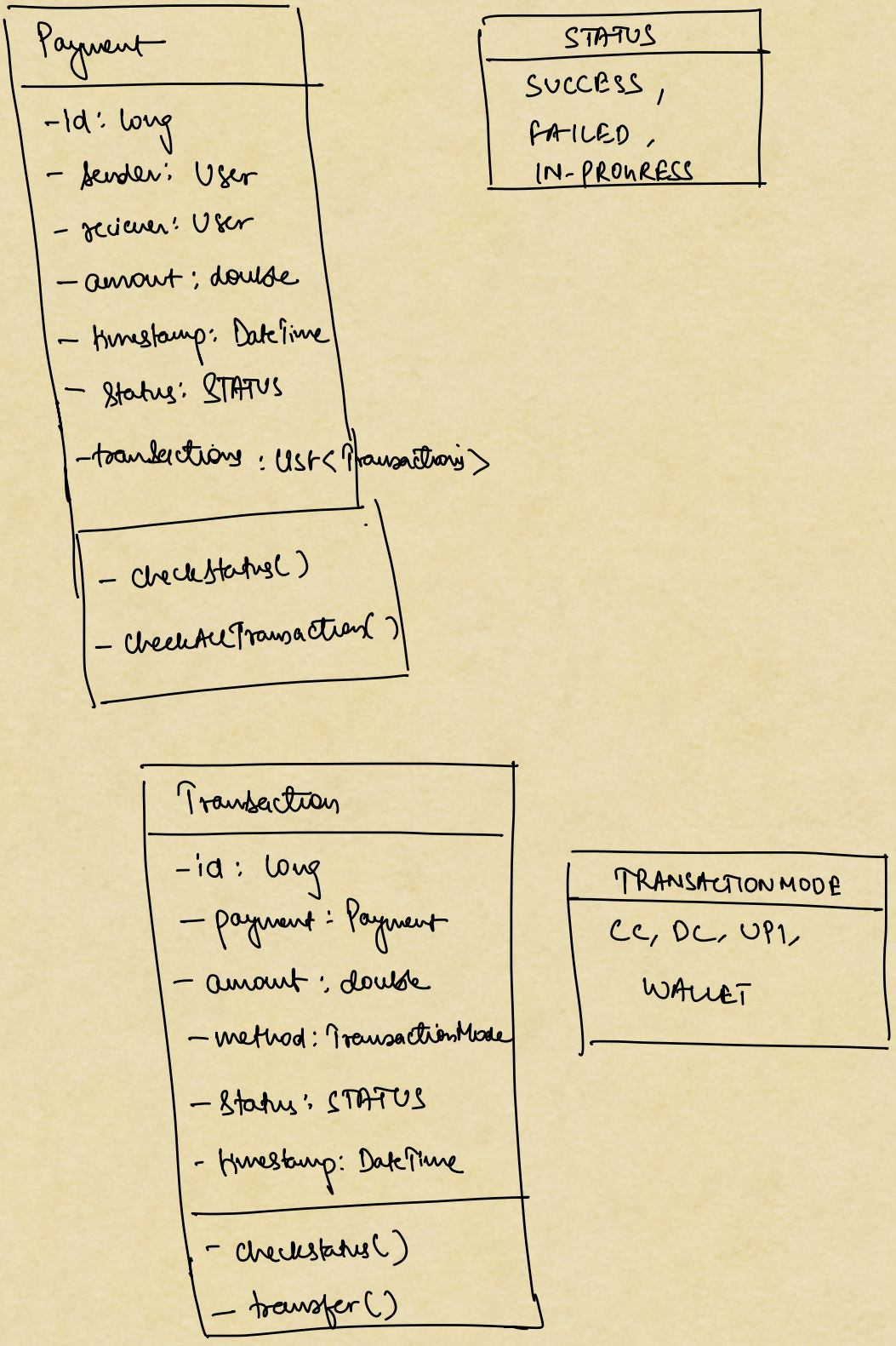
- VISUALISE USER JOURNEY

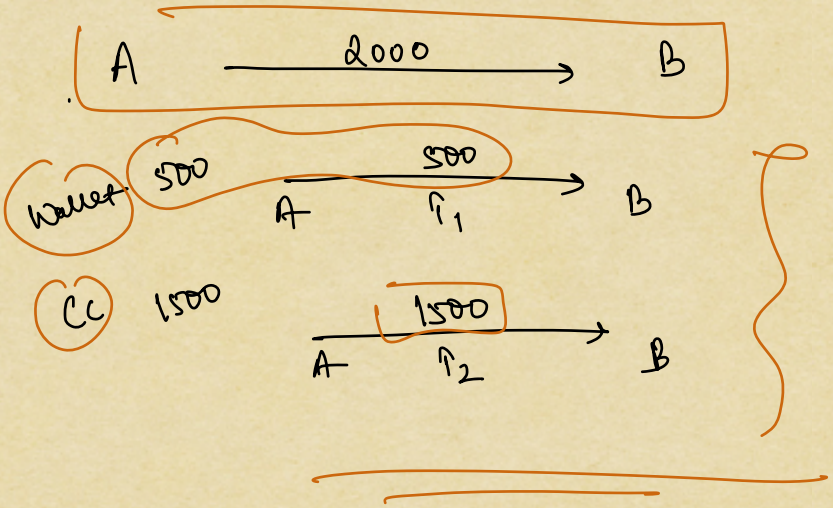


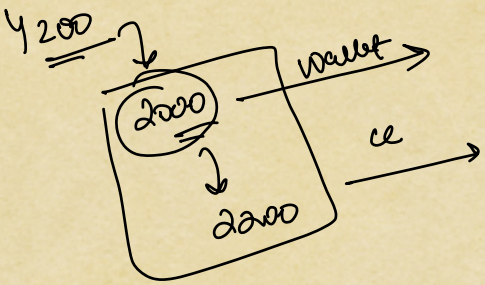
→ Entities for the payment app:-

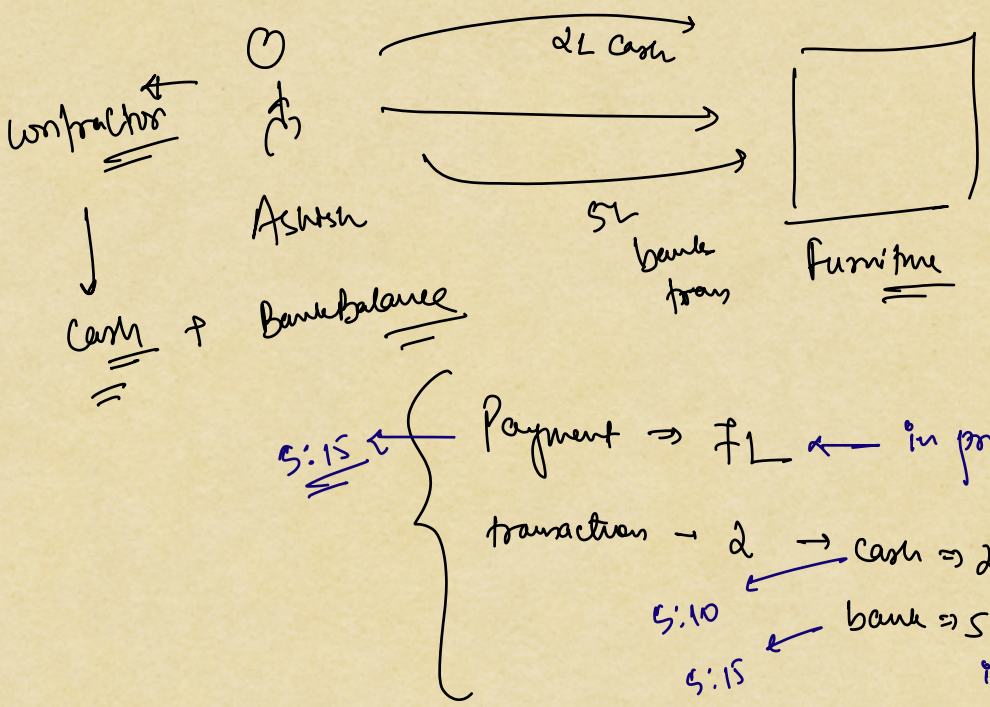
1) USER

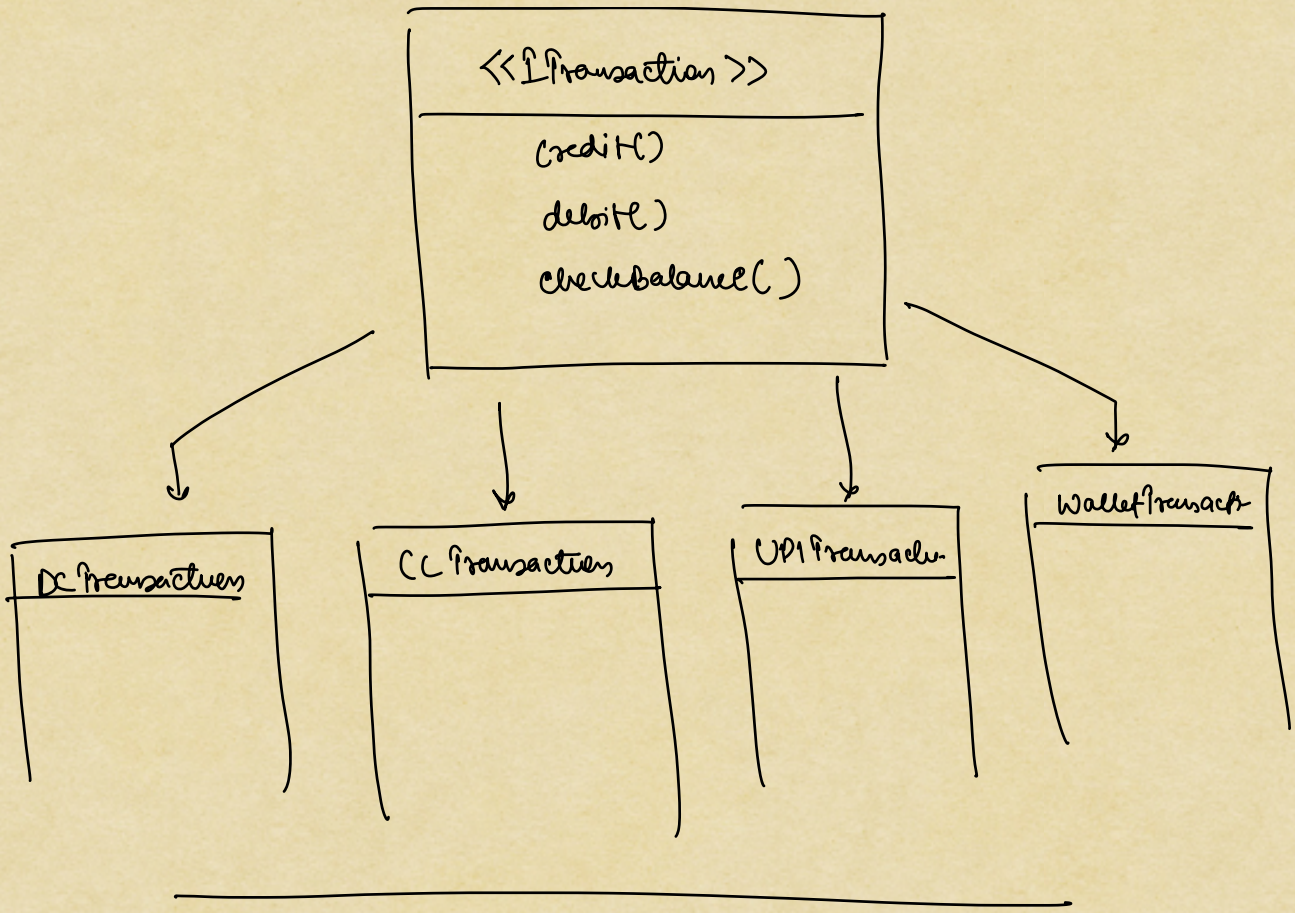



Payment
-id: long
- sender: User
- reciever: User
- amount: double
- timestamp: DateTime
- status: STATUS
-transactions: List<Transaction>
- checkStatus()
- checkAllTransaction()
STATUS
SUCCESS,
FAILED,
IN-PROGRESS
Transaction
-id: long
- payment: Payment
- amount: double
- method: TransactionMode
- status: STATUS
- timestamp: DateTime
- checkStatus()
- transfer()
TRANSACTIONMODE
CC, DC, UPI,
WALLET

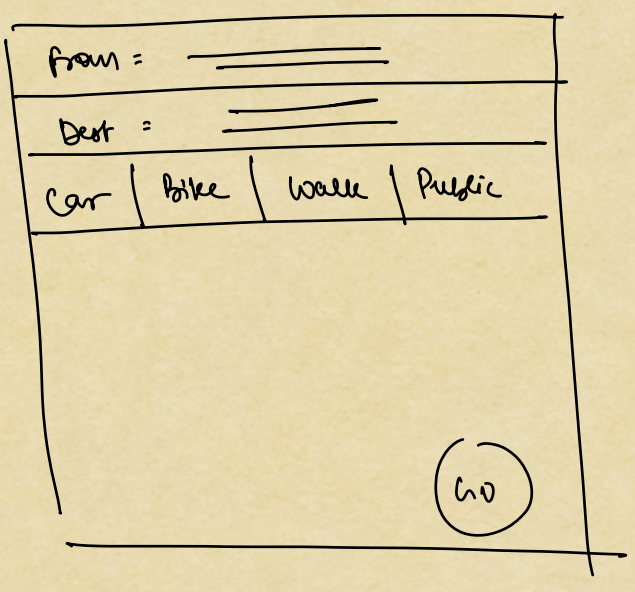

A
2000
B
wallet
500
500
A
T_1
B
CC
1500
1500
A
T_2
B


4200
2000
wallet
cc
2200

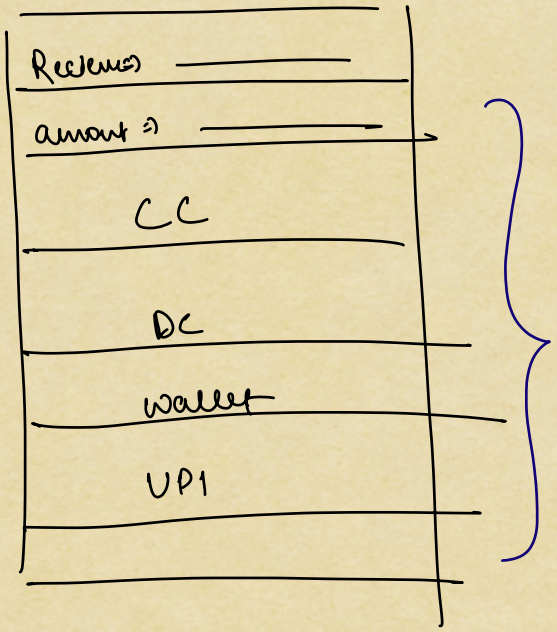

contractor
Cash + Bankbalance
Ashish
2L cash
5L
bank trans
Furniture
Payment ⇒ 7L ← in progress
5:15
transaction → 2 → cash ⇒ 2L
5:10
bank ⇒ 5L ←
5:15
in progress



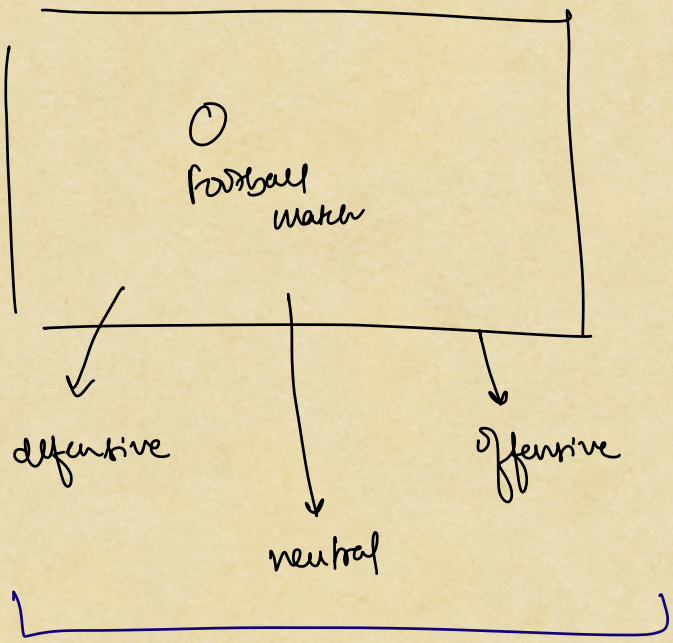
Strategy Design Pattern.



task is achieved via either mode of transport.



whatever mode we choose amount will be transferred to reciever



goal is winning the game. no matter which strategy I choose

All these are implemented and used using Strategy design pattern

→ When we will do payment, lots of payment Object and transaction Objects will be created

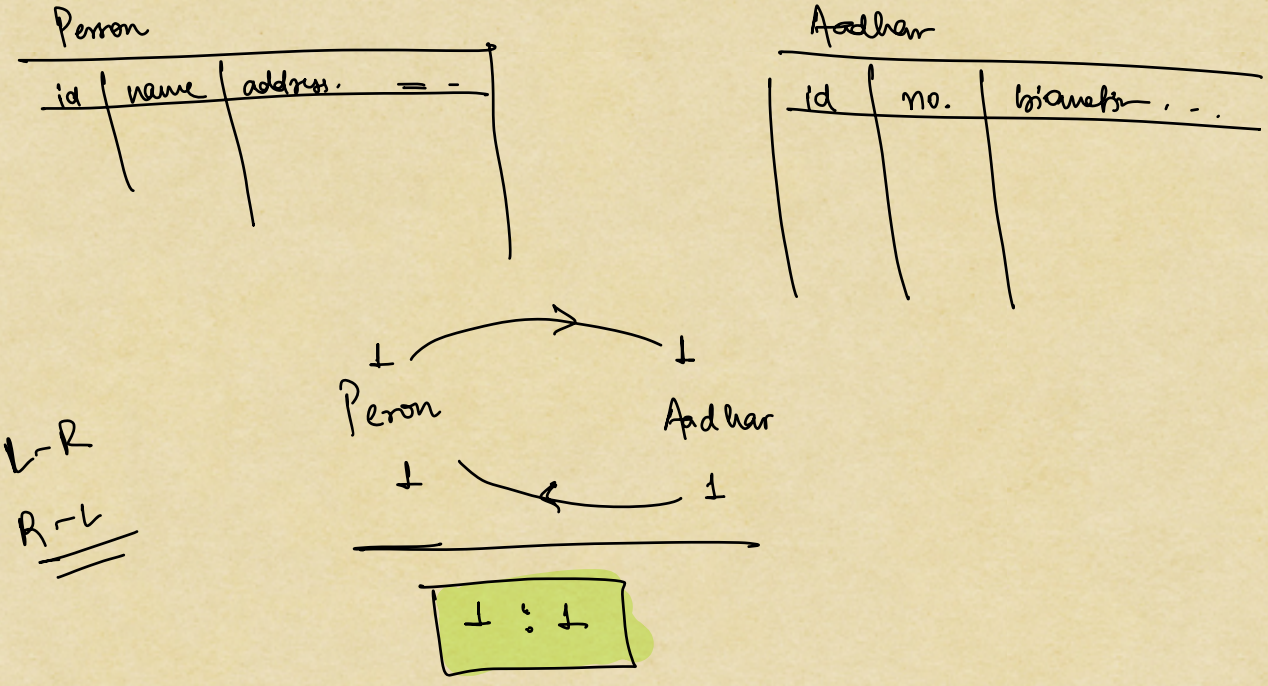
⇒ FACTORY DESIGN PATTERN

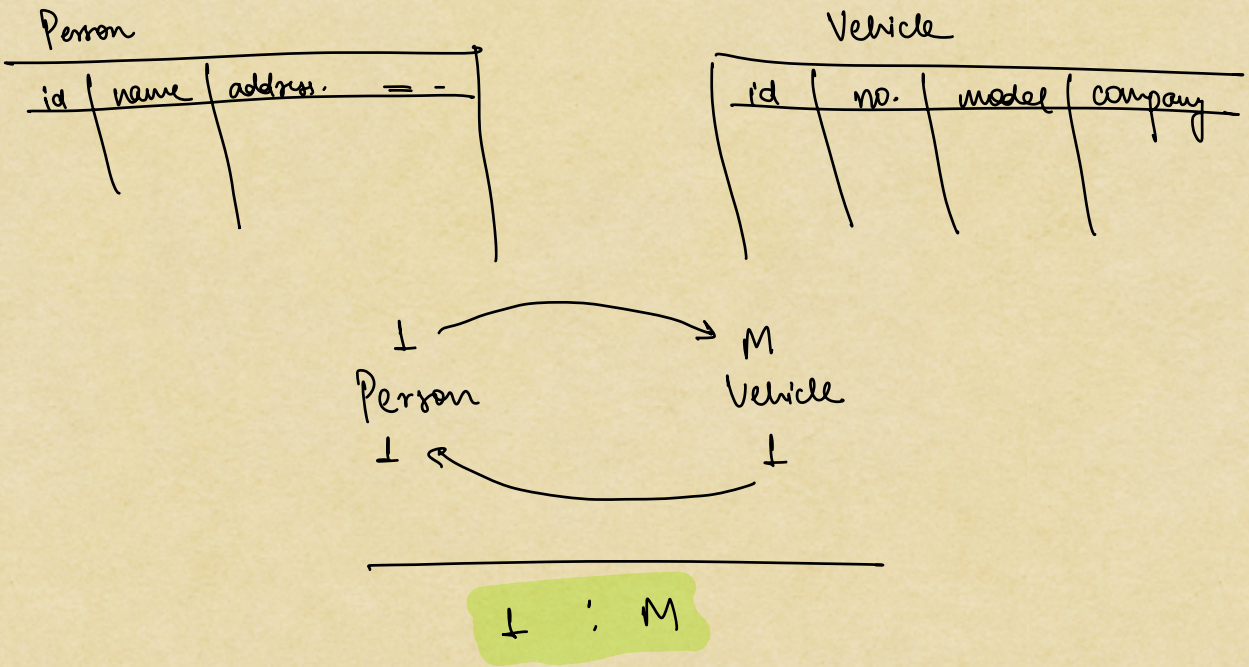
↓

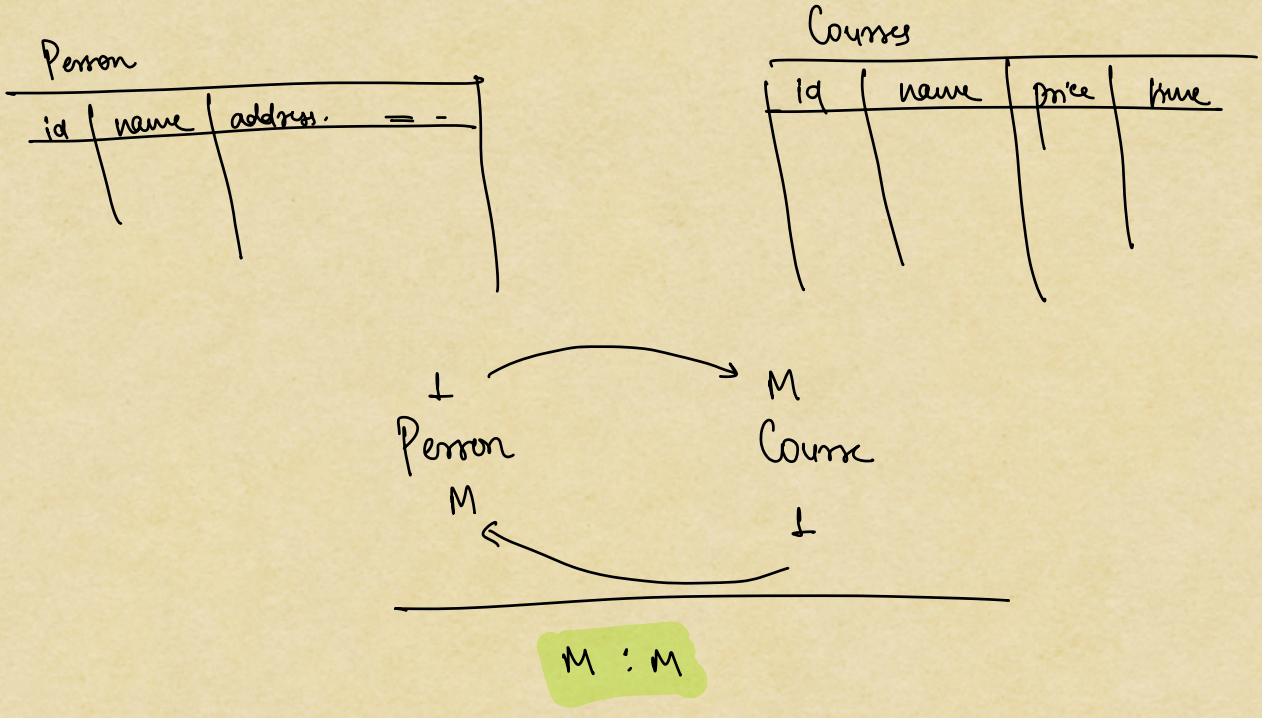
Solve the problem of creating similar Obj. multiple times

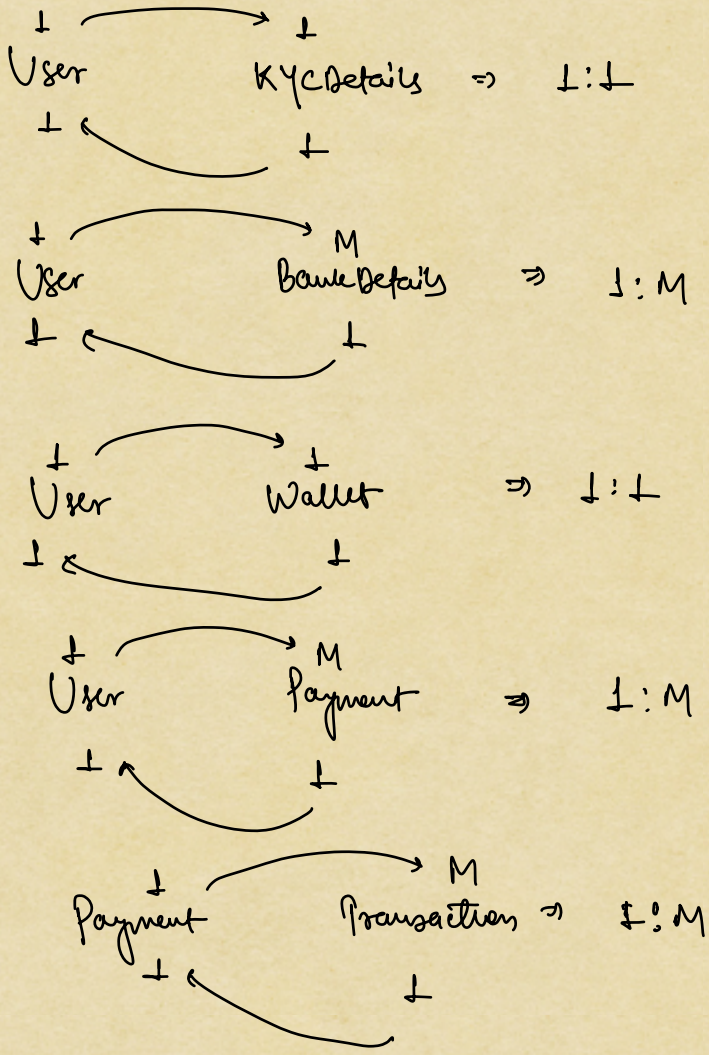
: CARDIANALITY :

1 : 1 | 1 : M | M : M




Person
id
name
address
Vehicle
id
no.
model
company
1
Person
1
M
Vehicle
1
1 : M


Person
id
name
address
Course
id
name
price
time
1
Person
M
M
Course
1
M : M



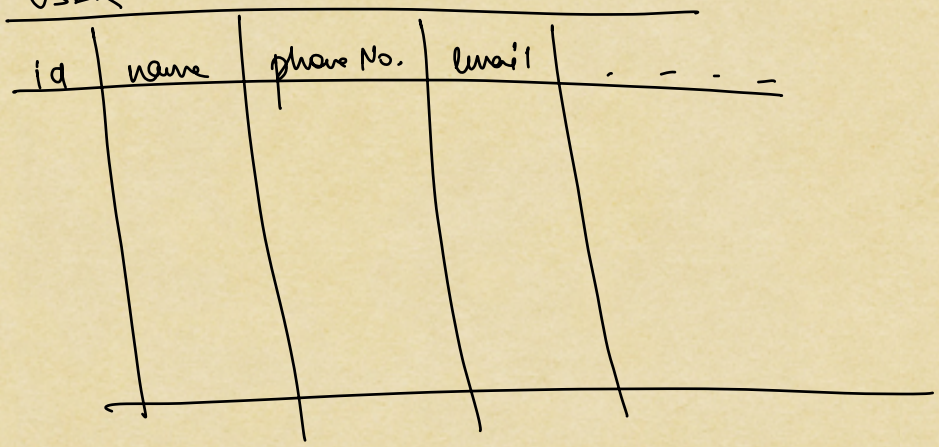
⇒ Schema Design:

i) Found all the entities of the system ✓

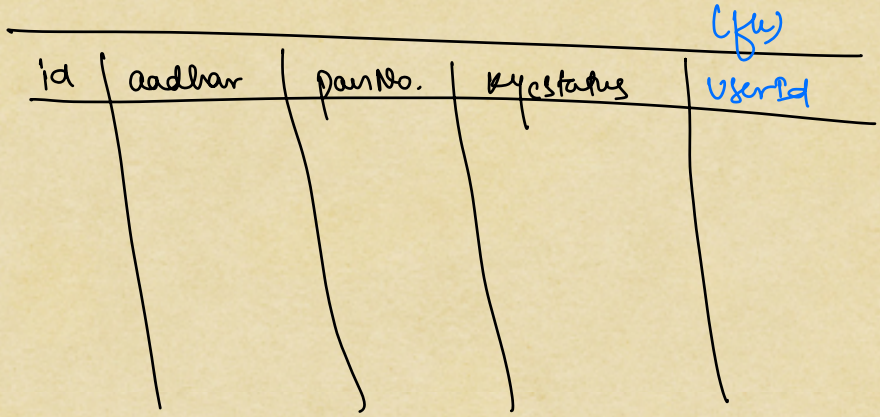
ii) Found all the cardinalities ✓

iii) How do you represent cardinality

USER



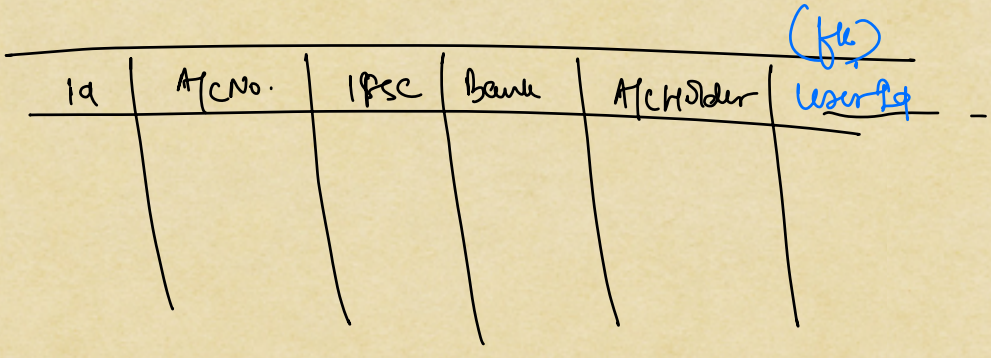
KYCDetails



1 : 1 ⇒ pk of 1 side ⟶ fk to other side

either way

BankDetails



1:M ⇒ pk of 1 side ⟶ fk to many side

User — Bank details

Wallet (fk)

id	amount	userId

Payment (fk)

id	amount	Status	timestamp	— —	userId

Transaction (fk)

id	amount	mode	status	. .	paymentId

⇒ Represent M:M ⇒ mapping table

Person

id	name	address
1	Shubham	
2	Ritwik	
3	Sujoy	

Courses

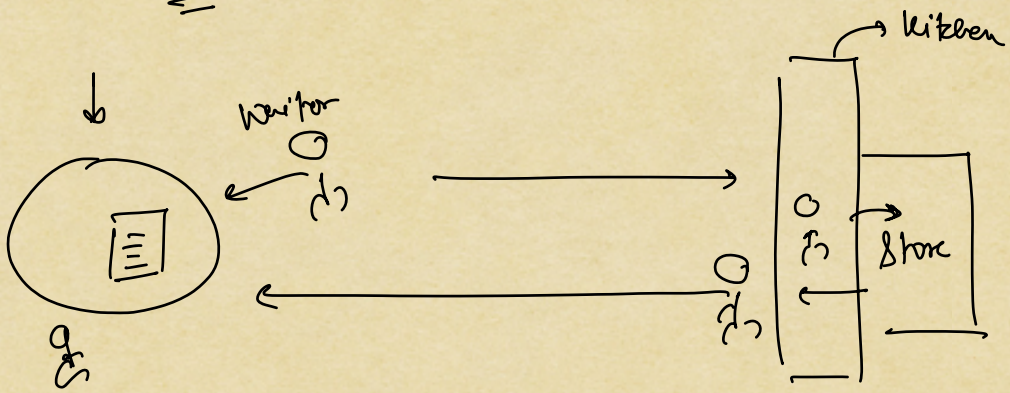
id	name	price	time
1	Dance		
2	Singing		
3	Music		
4			

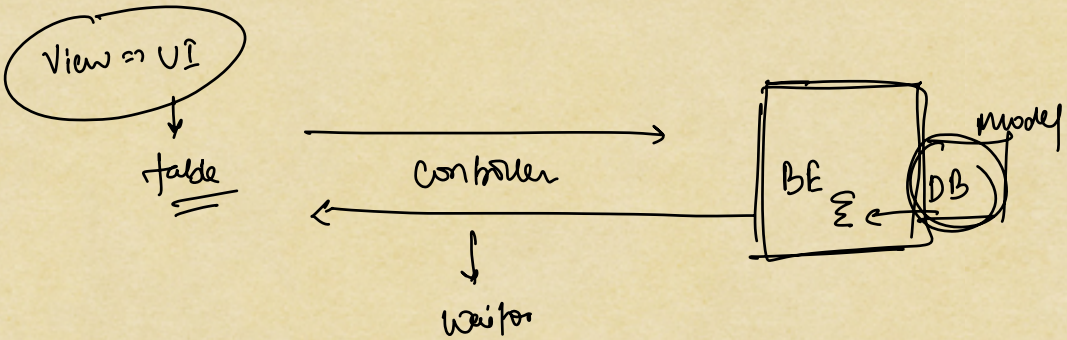
pe (fk) (fk)

personId	courseId
1	1
1	2
2	3
2	2
2	1
3	2
3	1
3	2
3	3

Bonus

MVC → model view controller




View = UI
Table
Controller
BE
DB
model
Waiter