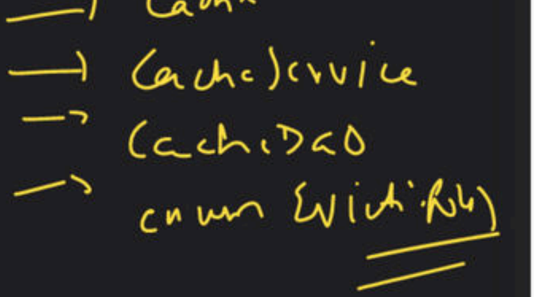
Solving Design Problems - Level II



Course on Solving Real World OOPs Design Problems

Love Babbar • Lesson 2 • June 5, 2022

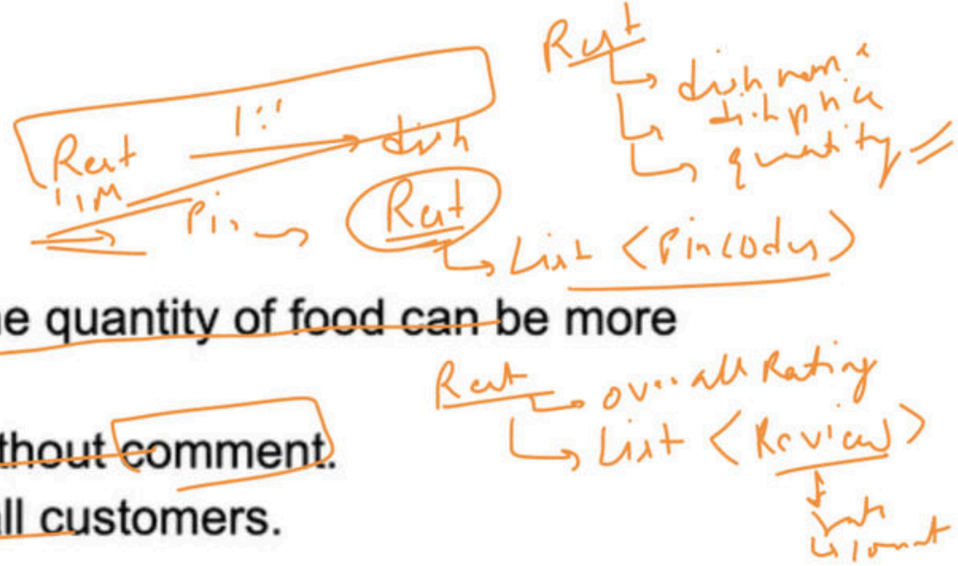
Foodkart:

Description:

Flapkart is starting a new online food ordering service. In this Service, users can order food from a restaurant which is serviceable in their area and the restaurant will deliver it.

Features:

1. Restaurants can only serve one specialized dish.
2. Restaurants can serve in multiple areas.
3. At a time, users can order from one restaurant, and the quantity of food can be more than one.
4. Users should be able to rate any restaurant with or without comment.
5. Rating of a restaurant is the average rating given by all customers.



Other Details:

1. Do not use any database or NoSQL store, use in-memory store for now.
2. Do not create any UI for the application.
3. Write a driver class for demo purposes. Which will execute all the commands at one place in the code and test cases.
4. Please prioritize code compilation, execution and completion.
5. Please do not access the internet for anything EXCEPT syntax.
6. You are free to use the language of your choice.
7. All work should be your own. If found otherwise, you may be disqualified.

Expectations:

1. Code should be demoable (very important)
2. Complete coding within the duration of 90 minutes.
3. Code should be modular, with Object Oriented design. Maintain good separation of concerns.
4. Code should be extensible. It should be easy to add/remove functionality without rewriting the entire codebase.
5. Code should handle edge cases properly and fail gracefully.
6. Code should be readable. Follow good coding practices:
7. Use intuitive variable names, function names, class names etc.
8. Indent code properly.
9. Once the code is complete please zip the source code and upload it to: https://docs.google.com/

Requirements:

1. Register a User:
    a. register_user(user_details)
    b. user_details: name, gender, phoneNumber(unique) and pincode.
2. Users should be able to login, and all the operations will happen in the context of that user. If another user logged in, the previous user will automatically be logged out.
    a. login_user(user_id): this should set the context for all the next operation to be done by this user.
3. Register a restaurant in context of login user:
    a. Register_restaurant(resturant_name, list of serviceable pin-codes, food item name, food item price, initial quantity).
4. Restaurant owners should be able to increase the quantity of the food item.
    a. update_quantity(restaurant name, quantity to Add)
5. Users should be able to rate(1(Lowest)-5(Highest)) any restaurant with or without comment.
    a. rate_restaurant(restaurant name, rating, comment)
6. User should be able to get list of all serviceable restaurant, food item name and price in descending order: show_restaurant(rating/price)
    a. Based on rating
    b. Based on Price

    NOTE: A restaurant is serviceable when it delivers to the user's pincode and has non-zero quantity of food item.
7. Place an order from any restaurant with any allowed quantity.
    a. place_order(restaurant name, quantity)

Bonus:

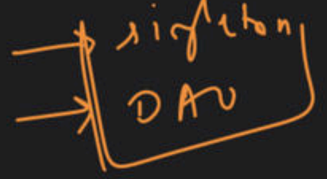
1. Order History of User: For a given user you should be able to fetch order history.

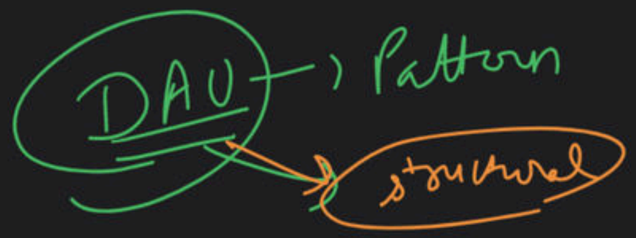
Order service

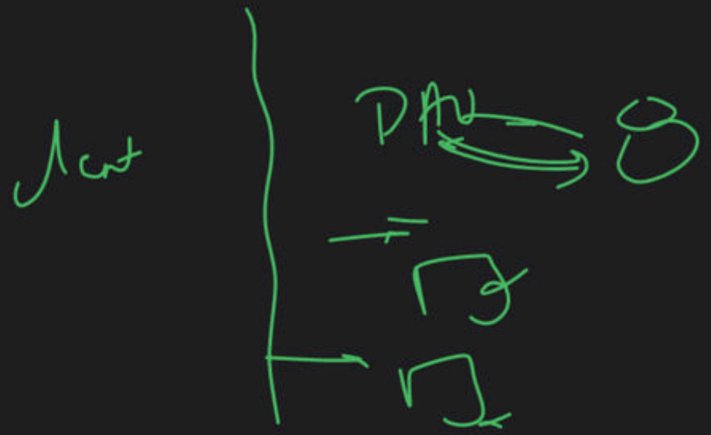
↳ place_Order ( )

↓

↳ List Order ( )







Order

↳ id

↳ User ID

↳ Restaurant ID

↳ item

↳ price

↳ timestamp

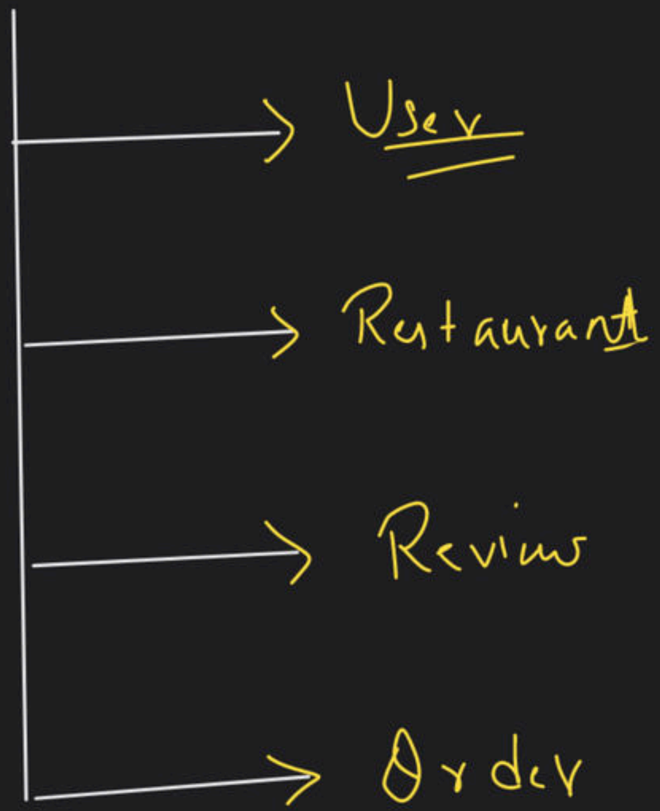
↳ quantity

↳ address

Principles

↳ DRY

Models



User :-

1. userID
2. name
3. phone
4. pincode
5. gender
6. List <Restaurant>
7. List <Order>

User.java

↳ D.M

↳ getter/setter

.enum Gender

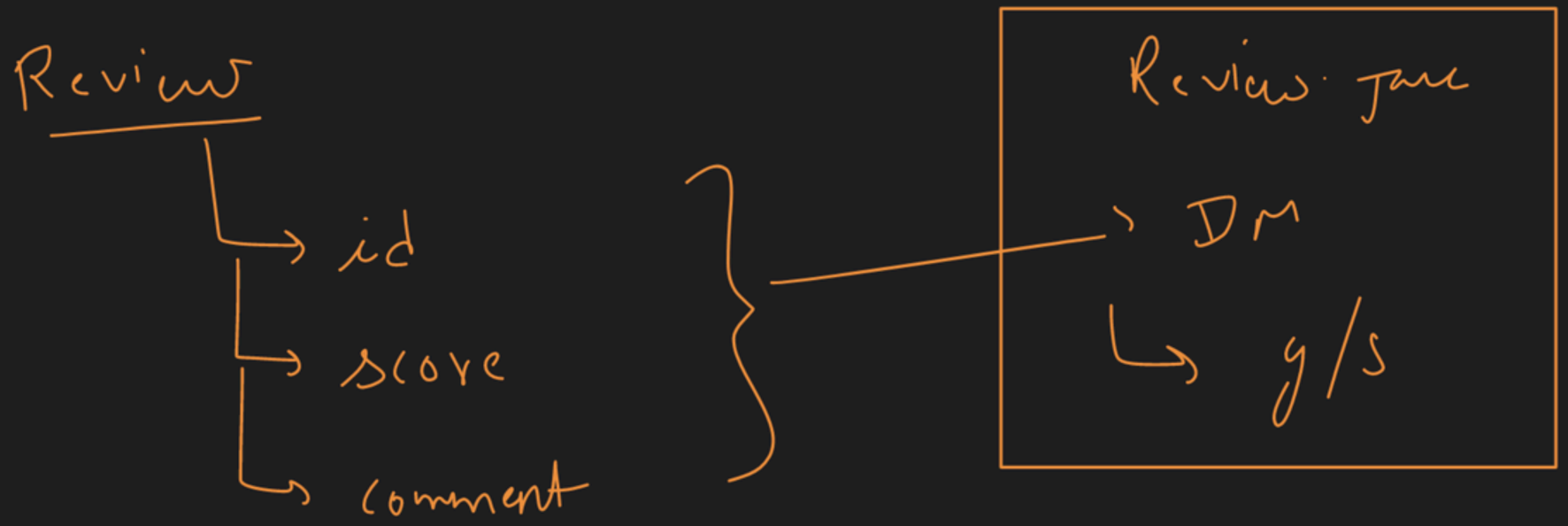
{

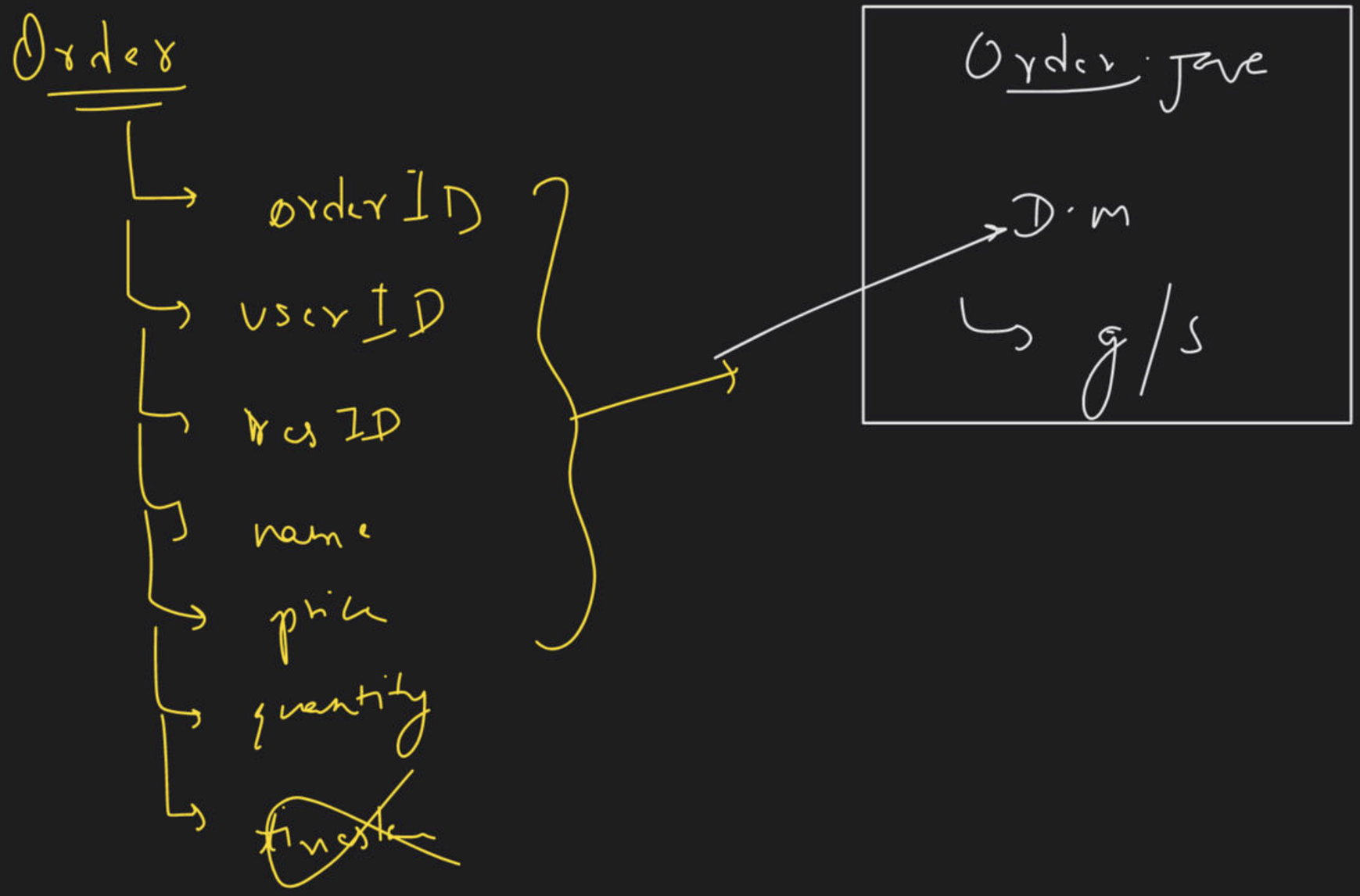
MALE

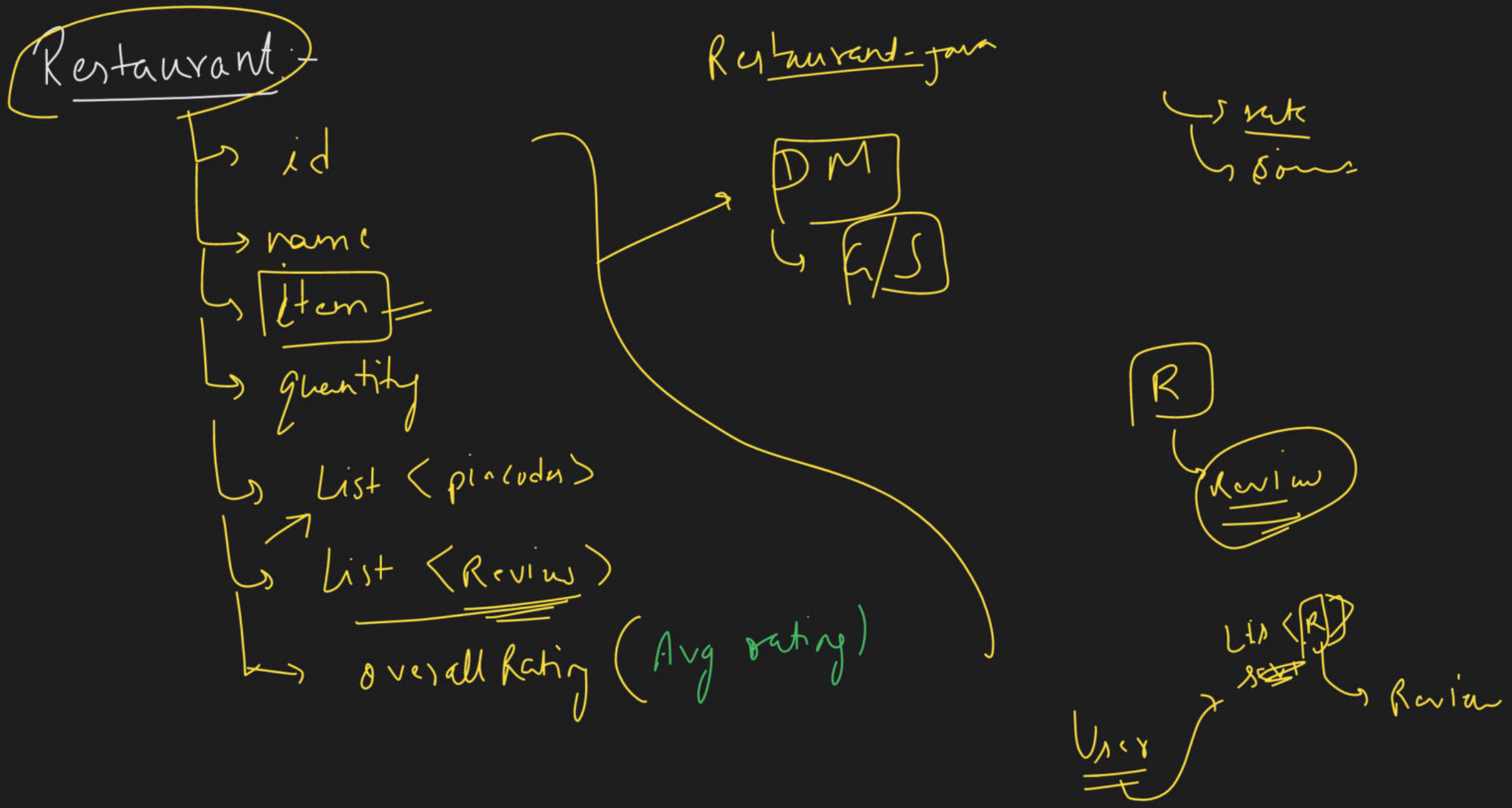
FEMALE

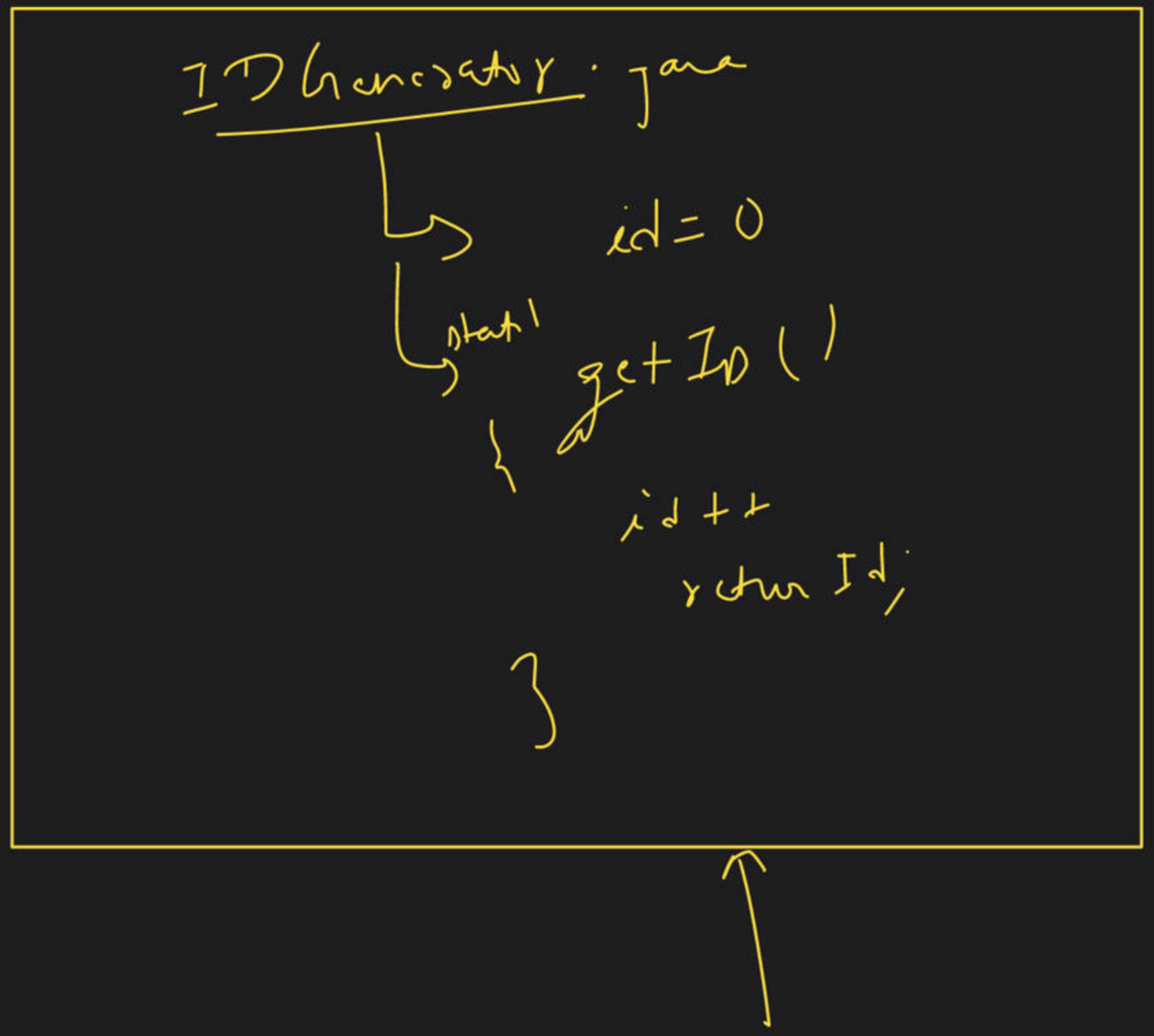
OTH

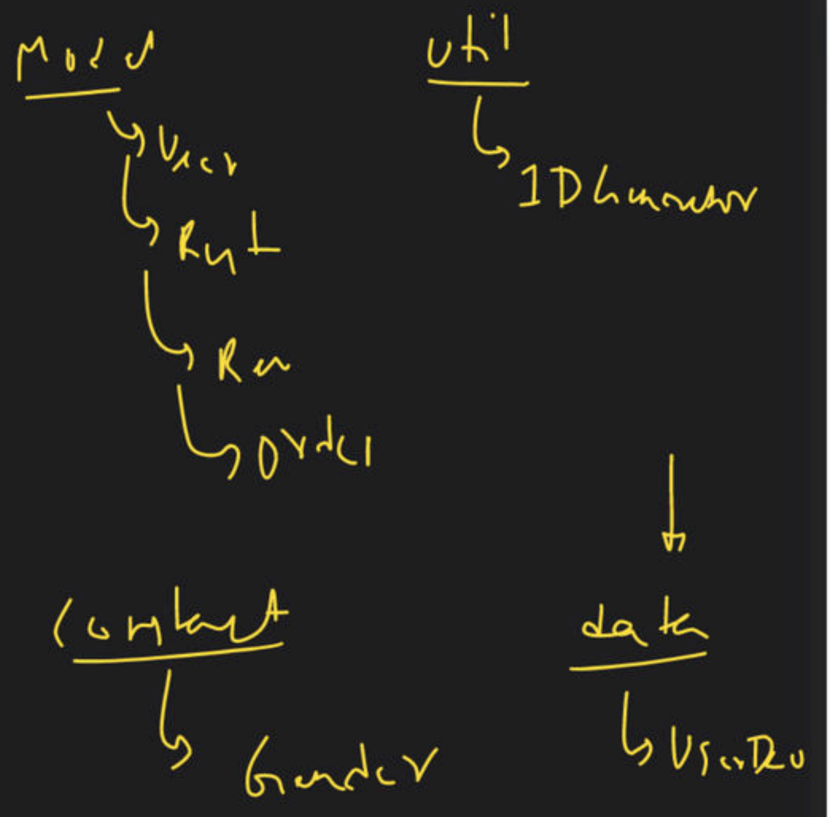
}


Review
id
score
comment
Review.java
DM
g/s

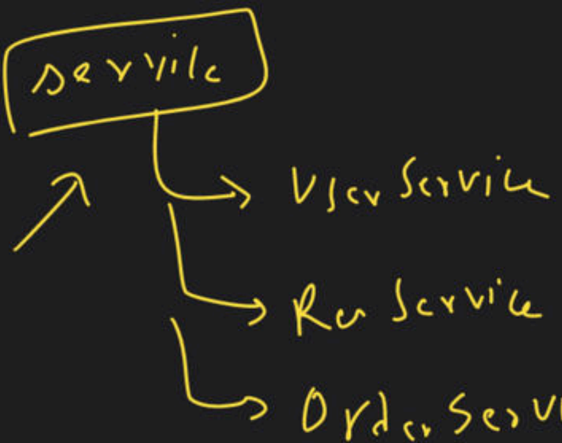

Order
orderID
userID
ProductID
name
price
quantity
finalPrice
Order.java
D.m
g/s

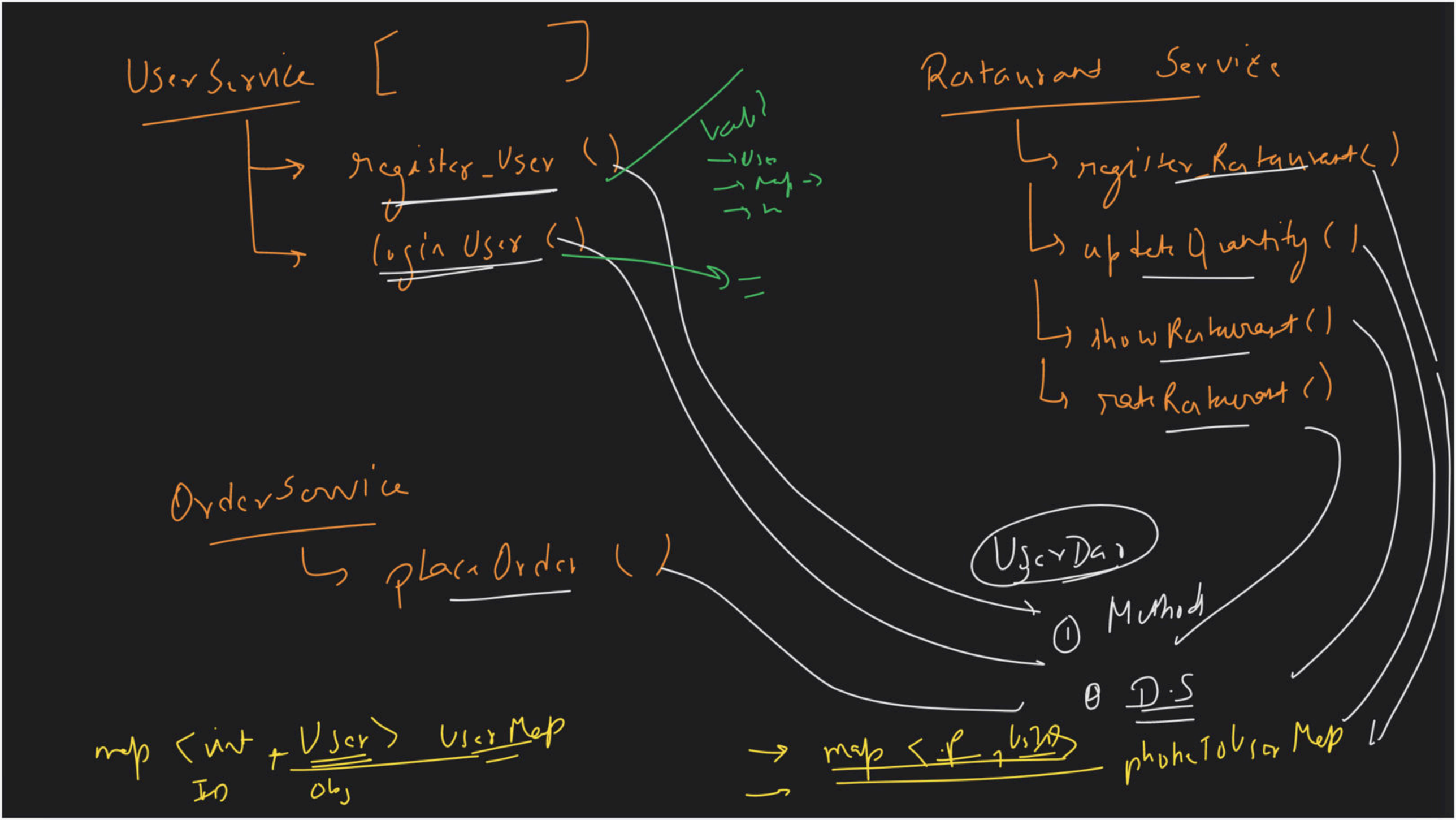

Restaurant
id
name
Item
quantity
List <picodes>
List <Review>
overall Rating (Avg rating)
Restaurant-java
DM
F/S
R
Review
User
List <R>

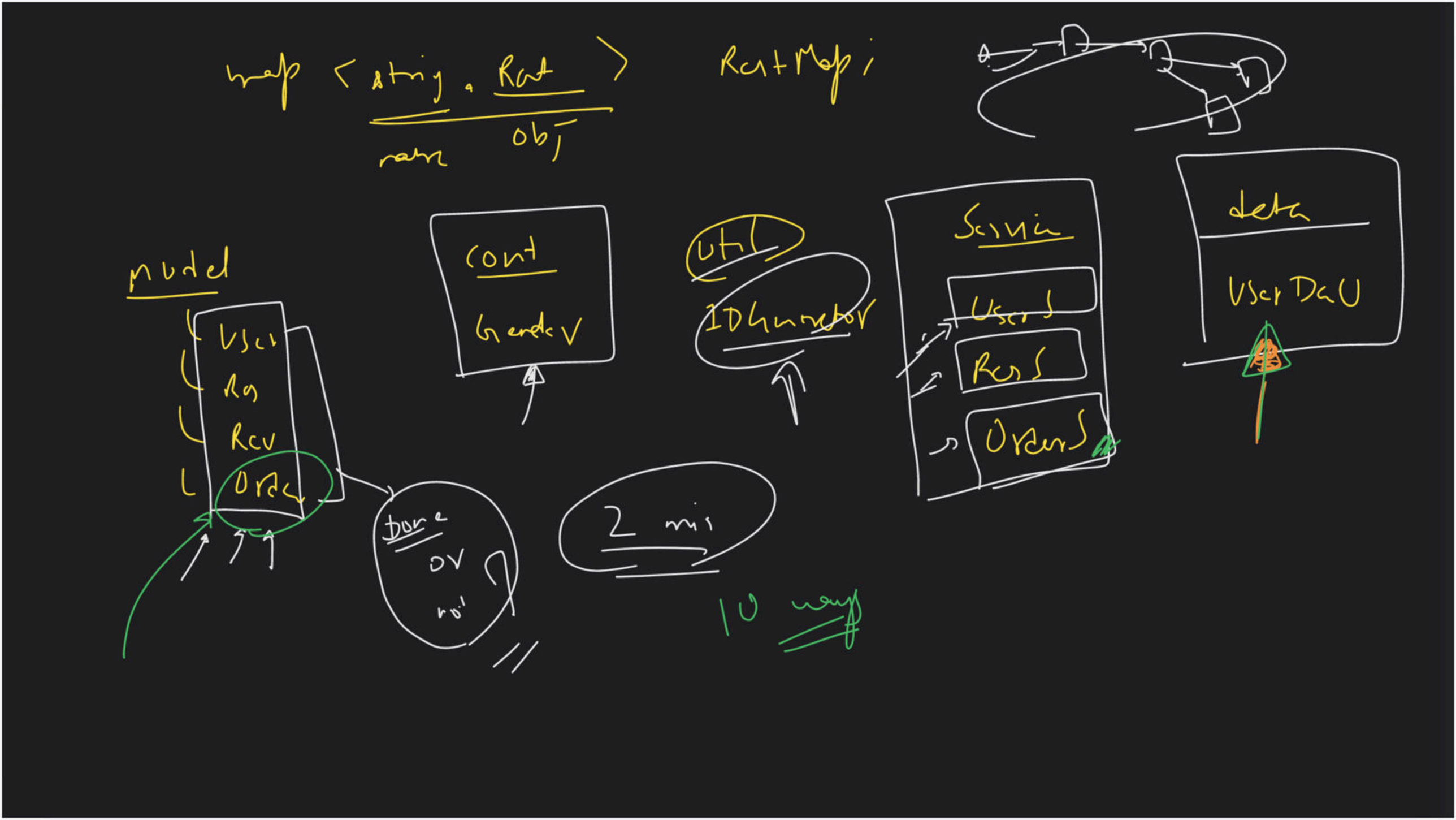

IDGenerator.java
id = 0
static
getID()
{
id++
return Id;
}

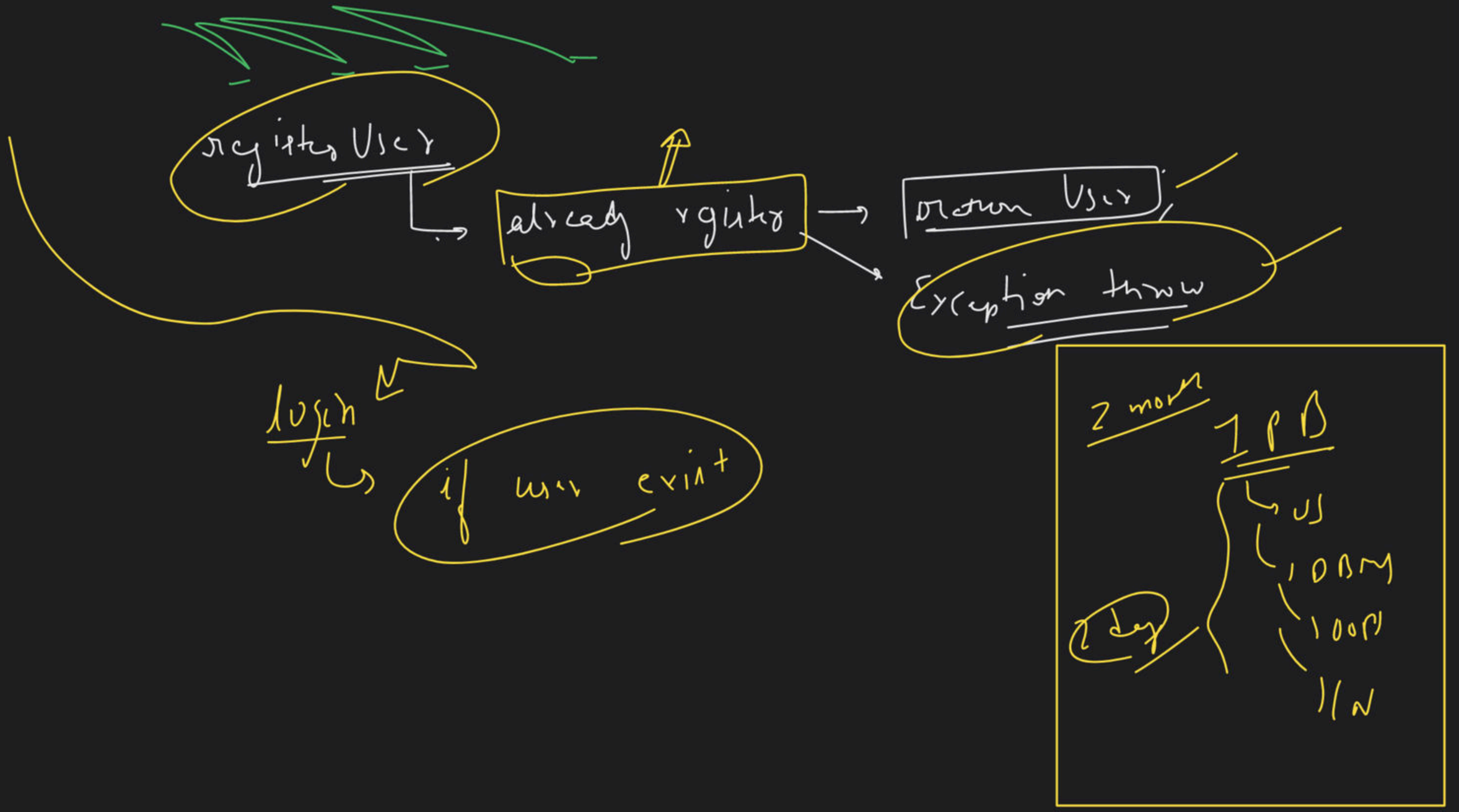

Model
User
Rent
Car
Order
util
IDGenerator
data
UserDto

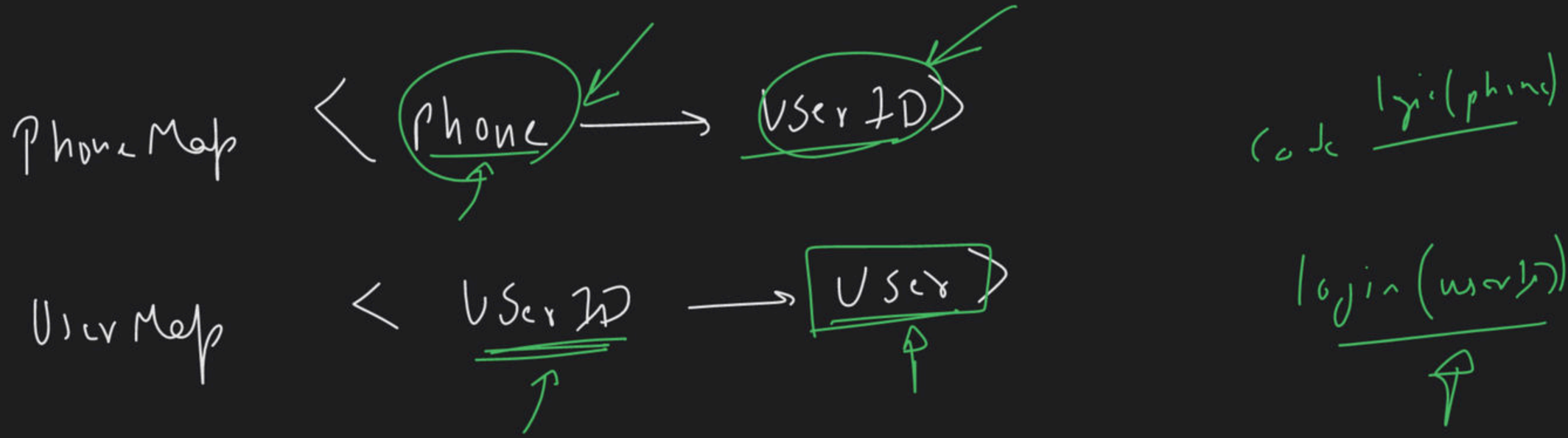
Gender


Service
UserService
RentService
OrderService

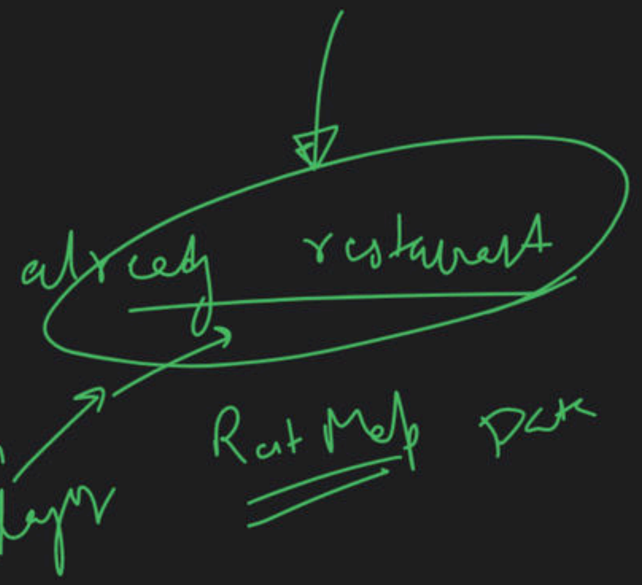

UserService
registerUser()
loginUser()
RestaurantService
registerRestaurant()
updateQuantity()
showRestaurant()
rateRestaurant()
OrderService
placeOrder()
UserDao
Method
DS
map<int, User> UserMap
map<if, User> phoneToUserMap

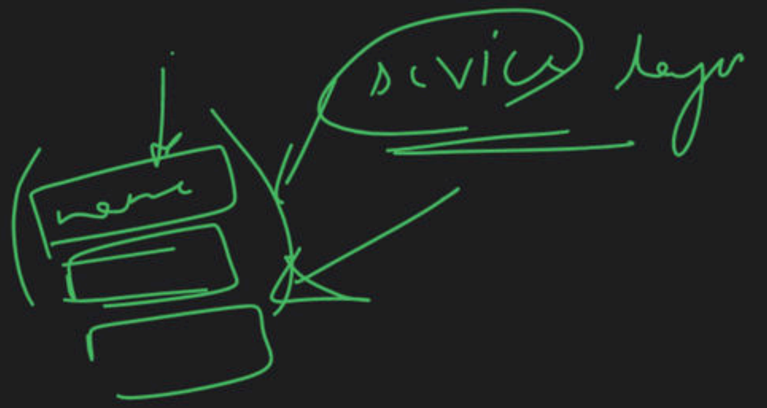

map<String, Rout>
RoutMap;
name
obj
model
User
Reg
Rev
Order
cont
Gender
util
IDGenerator
Servic
User
Rev
Order
data
UserDao
Date
2 mi
10 way


registering User
already registered
return User
Exception throw
login
if user exist
2 more
IPD
1 day


PhoneMap
Phone
UserID
User
UserMap
UserID
code
login(phone)
login(user)

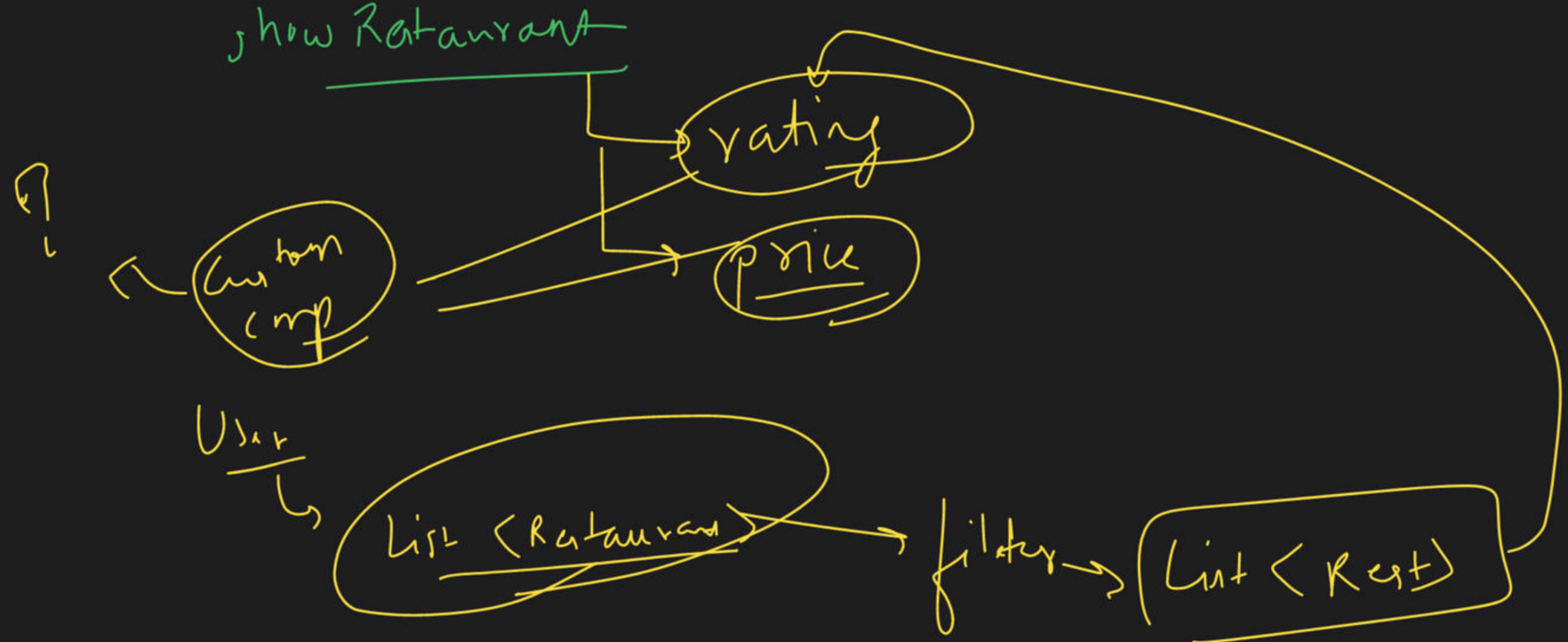
→ register Rest





If reviews → overallrating = rating

n^{th} reviews = $\frac{(\text{overall Rating} * \text{Total Reviews} + \text{rating})}{(\text{Total Reviews} + 1)}$


showRestaurant
rating
price
custom comp
User
List<Restaurant>
filter
List<Rest>

Driver class → ?