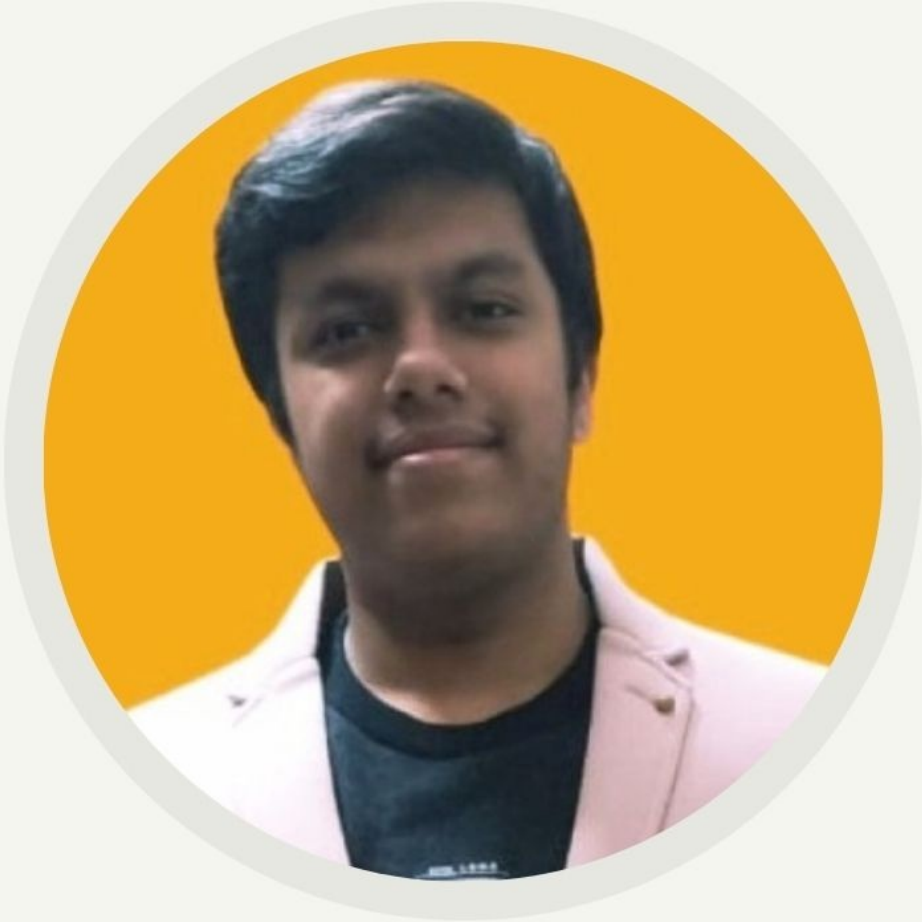
Free MasterClass

Design Splitwise

LOW LEVEL DESIGN | MACHINE CODING

GEEKSTER

BY ARCHIT AGGARWAL



About Me

Archit Aggarwal

- Software Engineer Intern at Salesforce
- 13 Months+ Experience in Edu-Tech
- 1000+ Students Mentored Live
- Full-Time Ex - DSA Instructor at Pepcoding
- Competitive Programmer

  (Expert at Codeforces)

Agenda

- → Importance of Low Level System Design or object Oriented Analysis & Design
- → Machine Coding Round Details & Problem Statement
- → oriented Oriented Design & it's Implementation
- → Follow ups or Further Exploration!

Introduction

Until recently, the general interview process for software engineers consisted of 2–3 rounds focused on problem-solving, data structures and algorithms, coupled with 1–2 rounds focused on checking the culture fit of the candidate. In the past few years, there has been a shift in approach with more and more companies adopting the machine coding round.

> Flipkart, Uber, Swiggy, Ola, Cred, etc. are some of the top tech companies where the first onsite round is the machine coding round.

What is machine coding round? and why is it important?

Format

Machine coding round involves solving a design problem in a matter of a couple of hours. It requires designing and coding a clean, modular and extensible solution based on a specific set of requirements. An example of a machine coding problem could be to design a snake and ladder game with certain requirements and constraints. This is generally followed by a code review process where an interviewer goes through the code and tries to understand the design decisions.

Expectations

Almost all the companies have similar judging criteria in the machine coding round. The expectations may vary based on seniority and also from company to company. These are the general expectations of a good solution:

- Code should be working and demonstrable.
- Code should be functionally correct.
- Code should be modular and readable.
- Separation of concerns should be addressed.
- Code should easily accommodate new requirements with minimal changes.
- There should be a main method from where the code could be easily testable.
- A UI is generally not required.

How to tackle machine coding round?

After getting the question (5-10 mins)

- **Read the problem statement carefully**. Try to clearly understand all the requirements.
- **Do not assume** anything that is not mentioned in the problem statement. If you want to make a specific assumption, discuss it with the interviewer at this stage.
- **Ask as many clarifying questions** as you can think of so as to make the requirements clear and remove any room for ambiguity or misinterpretation.

Getting to the solution (10-15 mins)

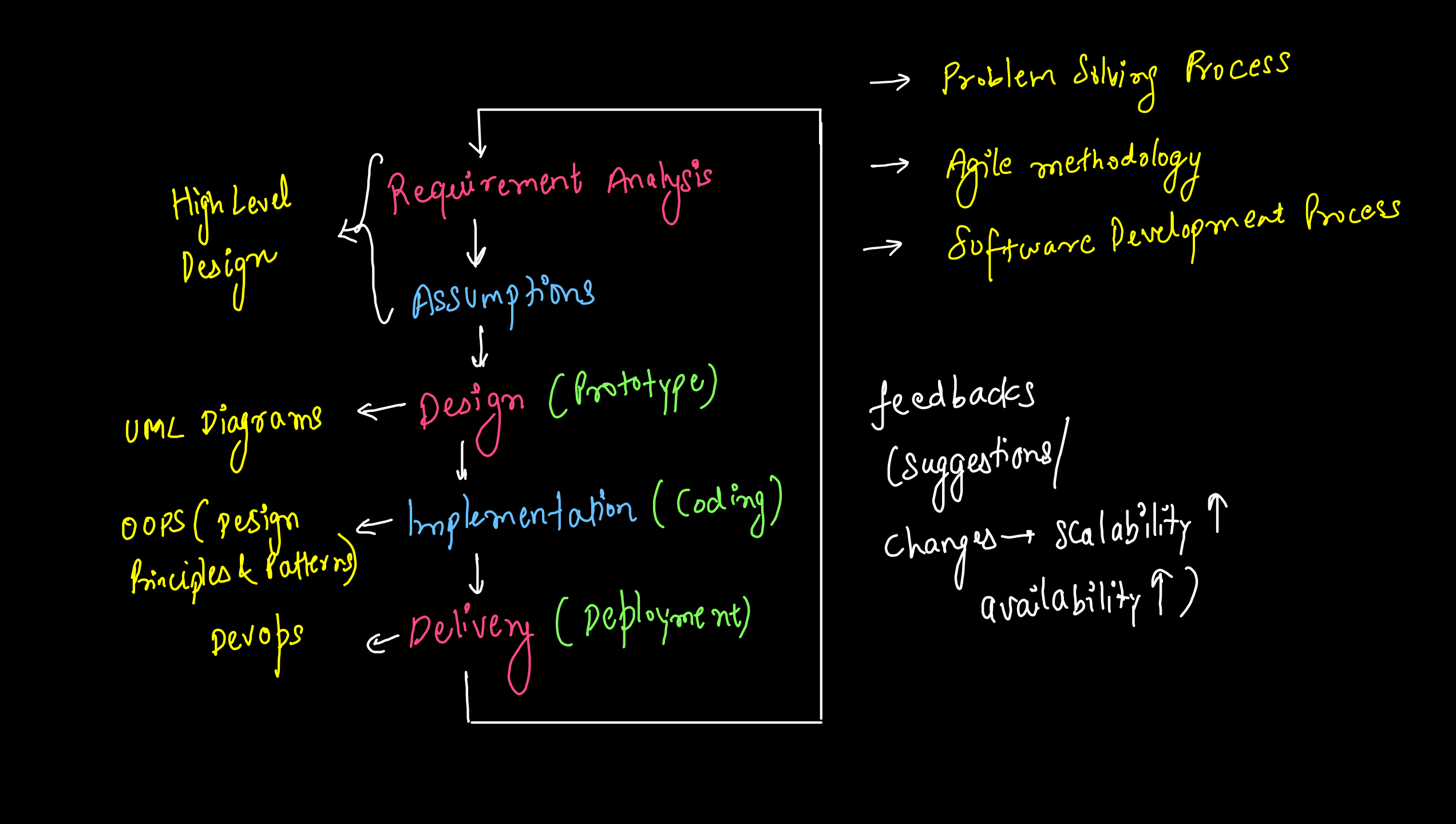
- Spend ~10 minutes **thinking about the design** of your solution. This is important. You do not want to start coding before a proper design.
- While designing think about how you can **make it extensible** to accommodate the optional requirements or any common extension that you can think of. Following good design principles and patterns will generally help.
- **Estimate how much time will it take you to code** with all the requirements. **Prioritize** which ones to do so as to at least solve the most critical ones.
- Apart from the design, estimate some time for creating the main method or API/CLI interface as mentioned in the problem statement or as clarified with the interviewer.
- Also, estimate some time for testing and making changes to have the solution working.
- If your current solution will take a lot of time to code, try to think of a simpler design that is good enough and will take less time to code. This is one of the biggest trade-offs in a machine coding round (and in general, software development).
- Do not prioritize an optional requirement over a mandatory requirement. If you focus more on extensibility and future requirements and are unable to complete a mandatory requirement, it's a red flag.
- Optional: If possible, draw a UML diagram of your design for clarity.

Coding (60-75 mins)

- If you've designed the solution and are comfortable in coding, this step should be fairly easy. Just make sure to **code fast** so as to complete as many requirements as possible.
- In the end, you need to have a **working code**. Be ready with good **sample examples** to demonstrate your solution.
- **Gracefully handle exceptions** and other corner cases. You do not want your code to fail during the demonstration.
- **Write readable code** with proper names. Use comments, if possible. You are writing the code for your interviewer to read and understand.
- Use a powerful IDE that you are comfortable with. Choose one where you can generate most of the boilerplate code to save time.

Demonstration

- While demonstrating, make sure to give a **high-level overview** of your solution. Do not explain each and every line of the code. Your code should be modular and self-explanatory.
- Tell the interviewer about all the requirements that you've completed and if your solution is extensible for other requirements.
- After running your solution on sample input, it may be a good idea to ask the interviewer if they want you to test with any other input.


→ Problem Solving Process
→ Agile methodology
→ Software Development Process
High Level Design
Requirement Analysis
Assumptions
Design (Prototype)
UML Diagrams
Implementation (Coding)
OOPS (Design Principles & Patterns)
Delivery (Deployment)
Devops
feedbacks (suggestions/ Changes → scalability ↑ availability ↑)

Problem Statement

Design an **Expense Sharing Application** where you can add your expenses, and split it among different people.

App keeps balances between people as in who owes how much to whom

Example

Consider 4 Friends (u1, u2, u3, u4) who goes to a trip to Manali.

At different events, different person pays for everyone.

Now, after the trip, they want to know, how much each needs to pay and to whom.

Mandatory

Requirements

- User: Each user should have a userId, name, email, mobile number.
- Expense: Could either be EQUAL, EXACT or PERCENT
- Users can add any amount, select any type of expense and split with any of the available users.
- The percent and amount provided could have decimals upto two decimal places.
- In case of percent, you need to verify if the total sum of percentage shares is 100 or not.
- In case of exact, you need to verify if the total sum of shares is equal to the total amount or not.
- The application should have a capability to show expenses for a single user as well as balances for everyone.
- When asked to show balances, the application should show balances of a user with all the users where there is a non-zero balance.
- The amount should be rounded off to two decimal places. Say if User1 paid 100 and amount is split equally among 3 people. Assign 33.34 to first person and 33.33 to others.

Input

- You can create a few users in your main method. No need to take it as input.
- There will be 3 types of input:
  - Expense in the format: `EXPENSE <user-id-of-person-who-paid> <no-of-users> <space-separated-list-of-users> <EQUAL/EXACT/PERCENT> <space-separated-values-in-case-of-non-equal>`
  - Show balances for all: `SHOW`
  - Show balances for a single user: `SHOW <user-id>`

Output

- When asked to show balance for a single user. Show all the balances that user is part of:
- Format: `<user-id-of-x> owes <user-id-of-y>: <amount>`
- If there are no balances for the input, print `No balances`
- In cases where the user for which balance was asked for, owes money, they'll be x. They'll be y otherwise.

Sample Output

```
No balances
No balances
User4 owes User1: 250
User2 owes User1: 250
User3 owes User1: 250
User4 owes User1: 250
User2 owes User1: 620
User3 owes User1: 1130
User4 owes User1: 250
User1 owes User4: 230
User2 owes User1: 620
User3 owes User1: 1130
User1 owes User4: 230
User2 owes User1: 620
User2 owes User4: 240
User3 owes User1: 1130
User3 owes User4: 240
```

Sample Input

```
SHOW
SHOW u1
EXPENSE u1 1000 4 u1 u2 u3 u4 EQUAL
SHOW u4
SHOW u1
EXPENSE u1 1250 2 u2 u3 EXACT 370 880
SHOW
EXPENSE u4 1200 4 u1 u2 u3 u4 PERCENT 40 20 20 20
SHOW u1
SHOW
```

u4 1200

4 u1 u2 u3 u4

40% 20% 20% 20%

480 240 240 240

u2 → u1 : 250 + 370

u3 → u1 : 250 + 880

u4 → u1 : 480 - 250

u2 → u4 : 240

u3 → u4 : 240

Expectations

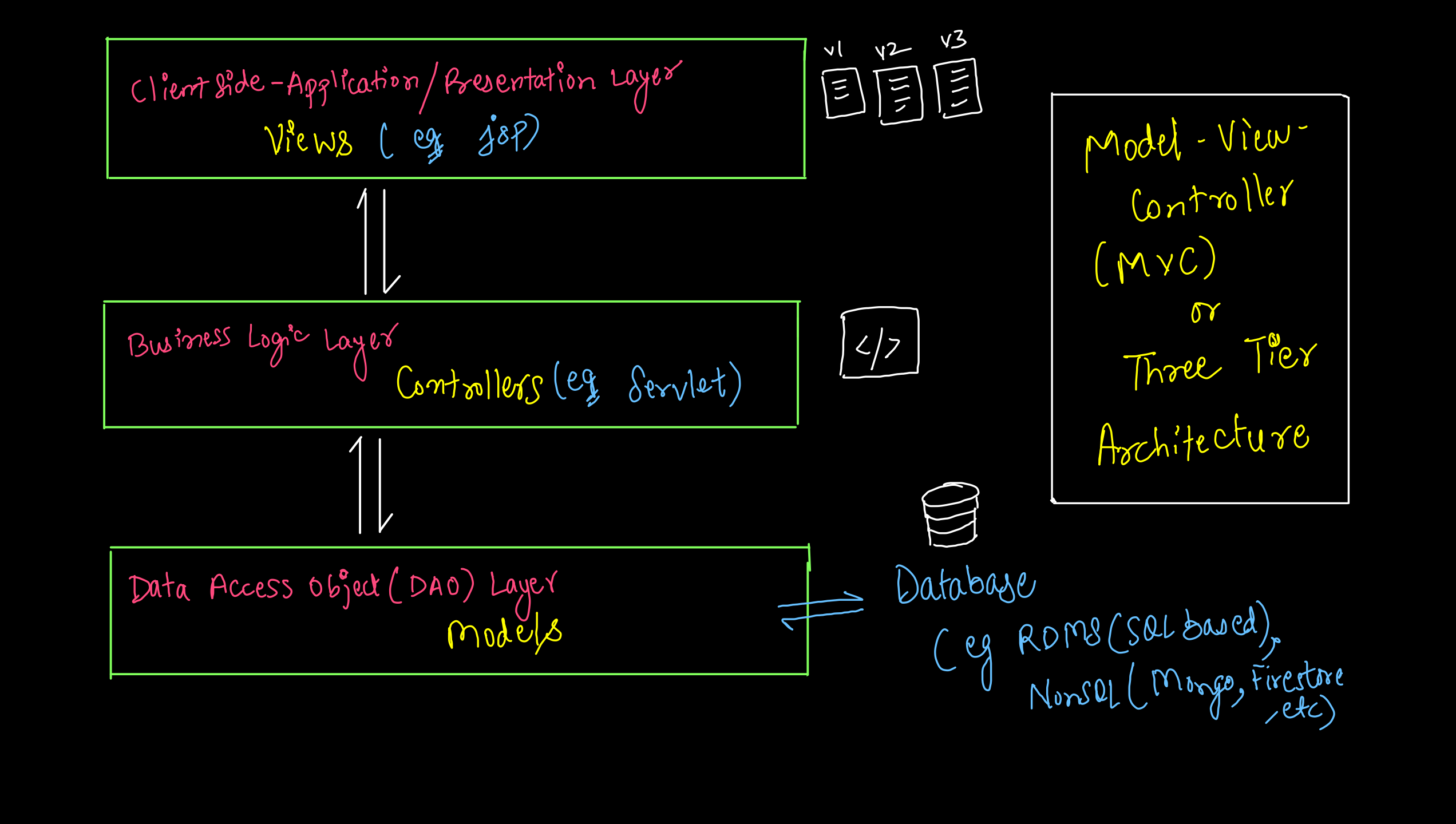
- Make sure that you have a working and demonstrable code
- Make sure that the code is functionally correct
- Code should be modular and readable
- Separation of concern should be addressed
- Please do not write everything in a single file
- Code should easily accommodate new requirements and minimal changes
- There should be a main method from where the code could be easily testable
- [Optional] Write unit tests, if possible
- No need to create a GUI

- In-memory database

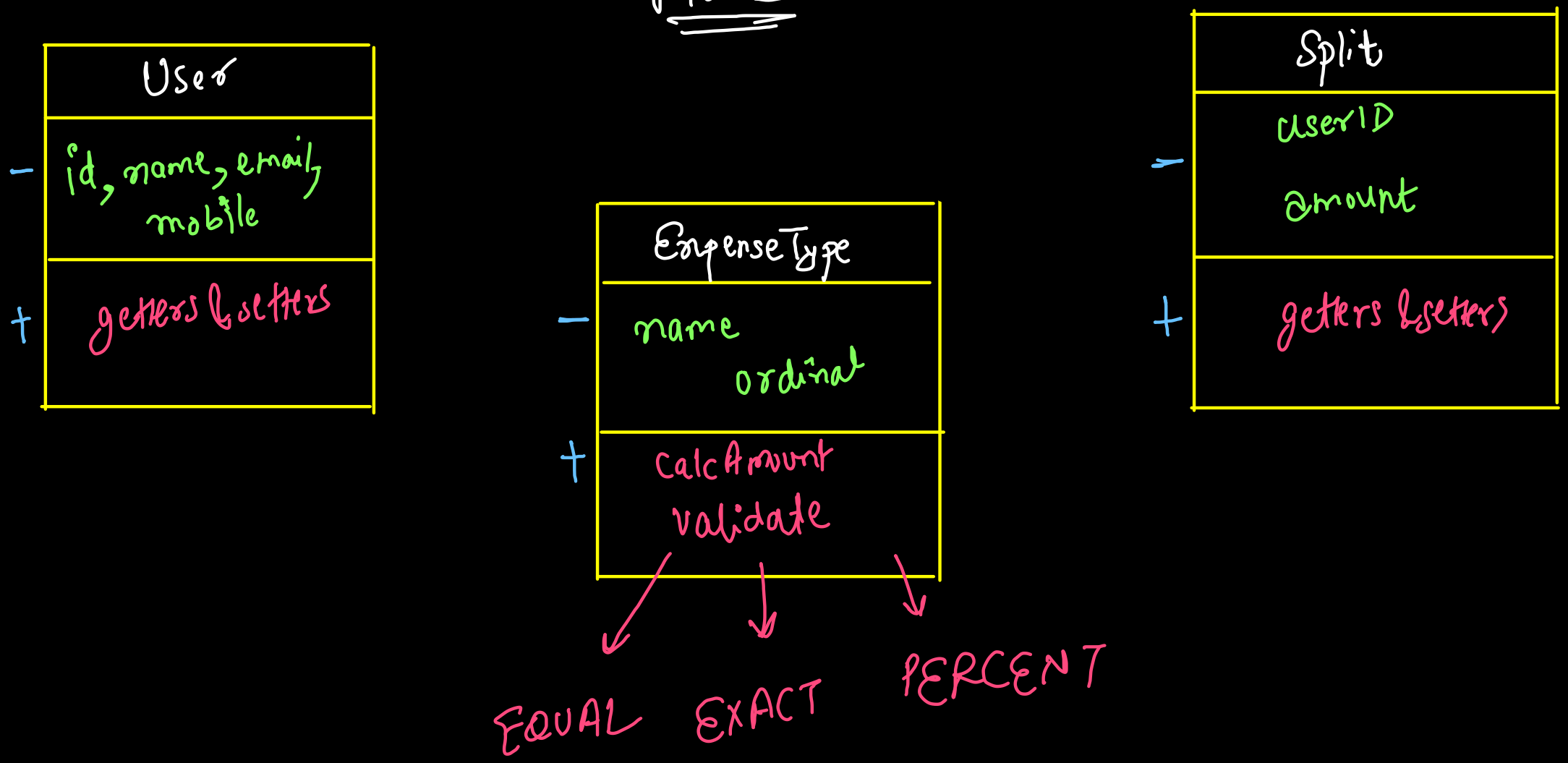
Optional Requirements

Please do these only if you've time left. You can write your code such that these could be accommodated without changing your code much.

- A way to add an expense name while adding the expense. Can also add notes, images, etc.
- Option to split by share. Ex: 'User4 pays and everyone splits equally. You pay for 2 people.' could be added as: `u4 1200 4 u1 u2 u3 u4 SHARE 2 1 1 1`
- A way to show the passbook for a user. The entries should show all the transactions a user was part of. You can print in any format you like.
- There can be an option to simplify expenses. When simplify expenses is turned on (is true), the balances should get simplified. Ex: 'User1 owes 250 to User2 and User2 owes 200 to User3' should simplify to 'User1 owes 50 to User2 and 200 to User3'.


Client side - Application / Presentation Layer
Views (eg jsp)
v1
v2
v3
Model - View - Controller (MVC) or Three Tier Architecture
Business Logic Layer
Controllers (eg Servlet)
</>
Data Access Object (DAO) Layer
Models
Database
(eg RDMS (SQL based), NonSQL (Mongo, Firestore, etc)

Models


User
- id, name, email, mobile
+ getters & setters
ExpenseType
- name
ordinal
+ calcAmount
validate
EQUAL
EXACT
PERCENT
Split
- userID
amount
+ getters & setters

Dao class

Map<String, Map<String, Double>> balances;

creditorID

debtorID

U2 → U1 : 250 + 370

U3 → U1 : 250 + 880

U1 → U4 : 480 - 250

U2 → U4 : 240

U3 → U4 : 240

U1 : [U2 : 250 + 370]
[U3 : 250 + 880]

U4 : [U1 : 480 - 250]
[U2 : 240]
[U3 : 240]

U2 : [ ] U3 : [ ] U4 : [