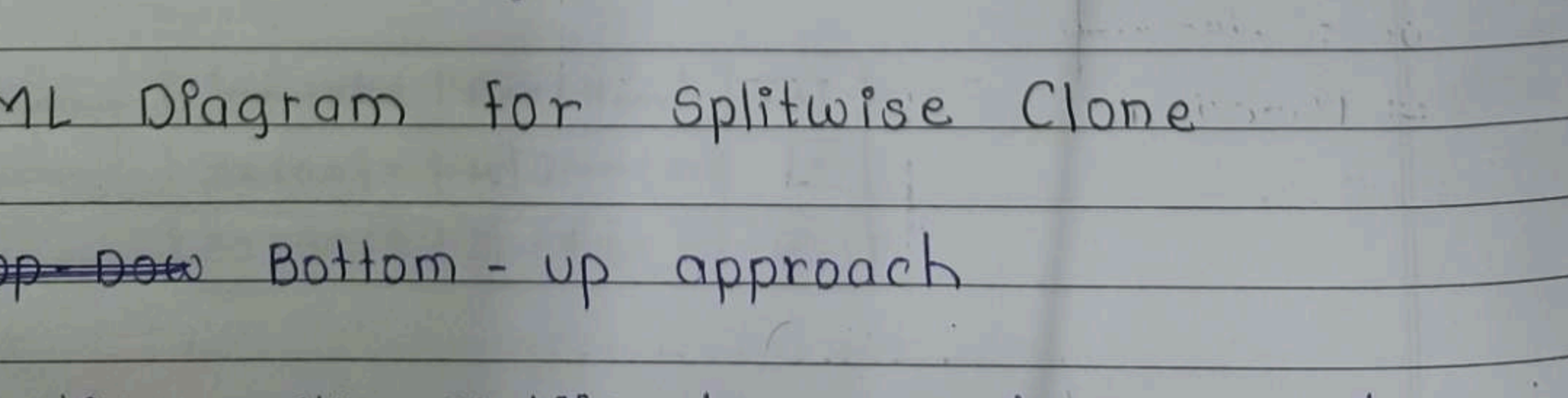
Lecture 31 : Splitwise Clone

Requirements

- → User can join or leave the group
- → User can add expense in a group
- → User can settle an expense in a group
- → Adding expense should be based on several strategies (equal split, % split, exact split)
- → User can't leave group without ~~settle~~ setteling expenses
- → User can also add individuals expenses, outside group (one-on-one)
- → A notification to be sent when an expense is added / settled.

Happy Flow



- Simplify Transaction
  - ↳ implement using Greedy Algorithm
  - ↳ add on feature to reduce no. of transactions overall in group (detail at end)

UML Diagram for Splitwise Clone

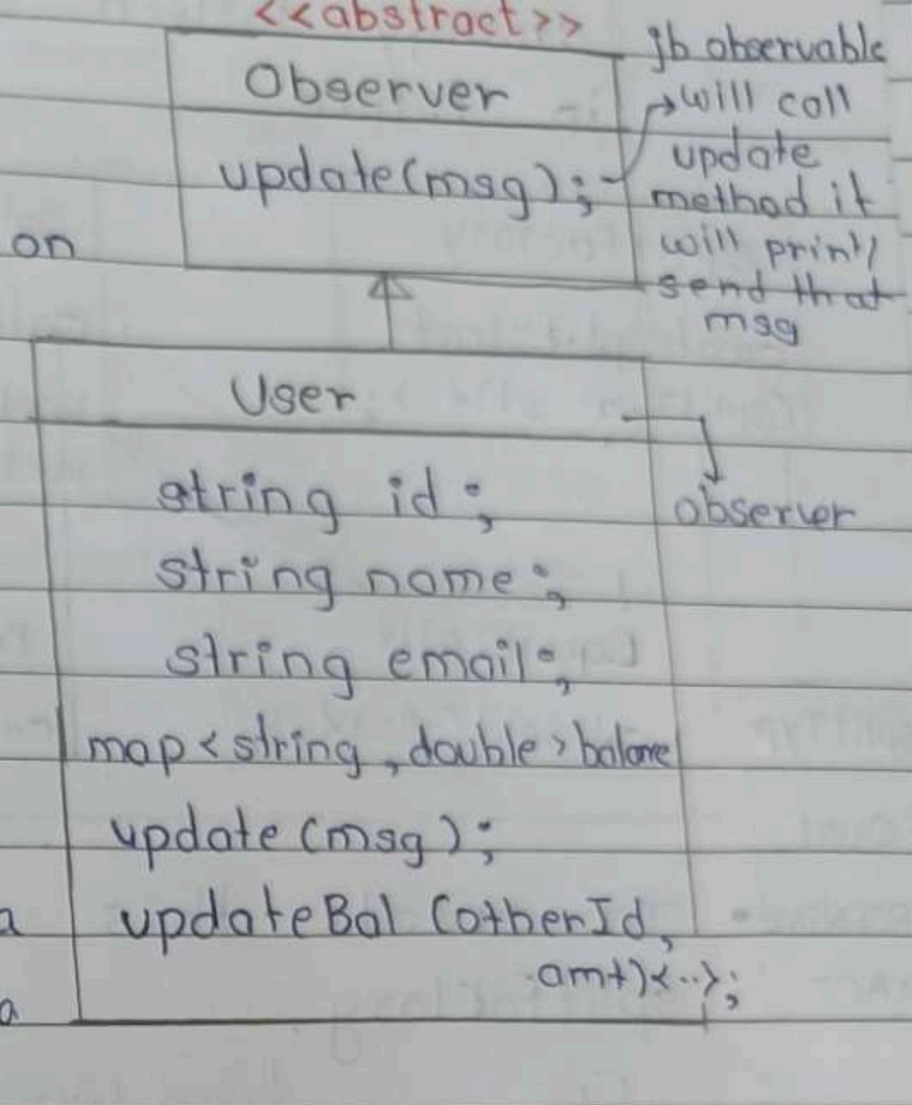
- ~~Top-Down~~ Bottom-up approach
- Starting with Notification System : made using Observer Pattern

- Humare pass 2 options hai for observable i.e.
  - Groups (✓) we prefer this
  - Splitwise

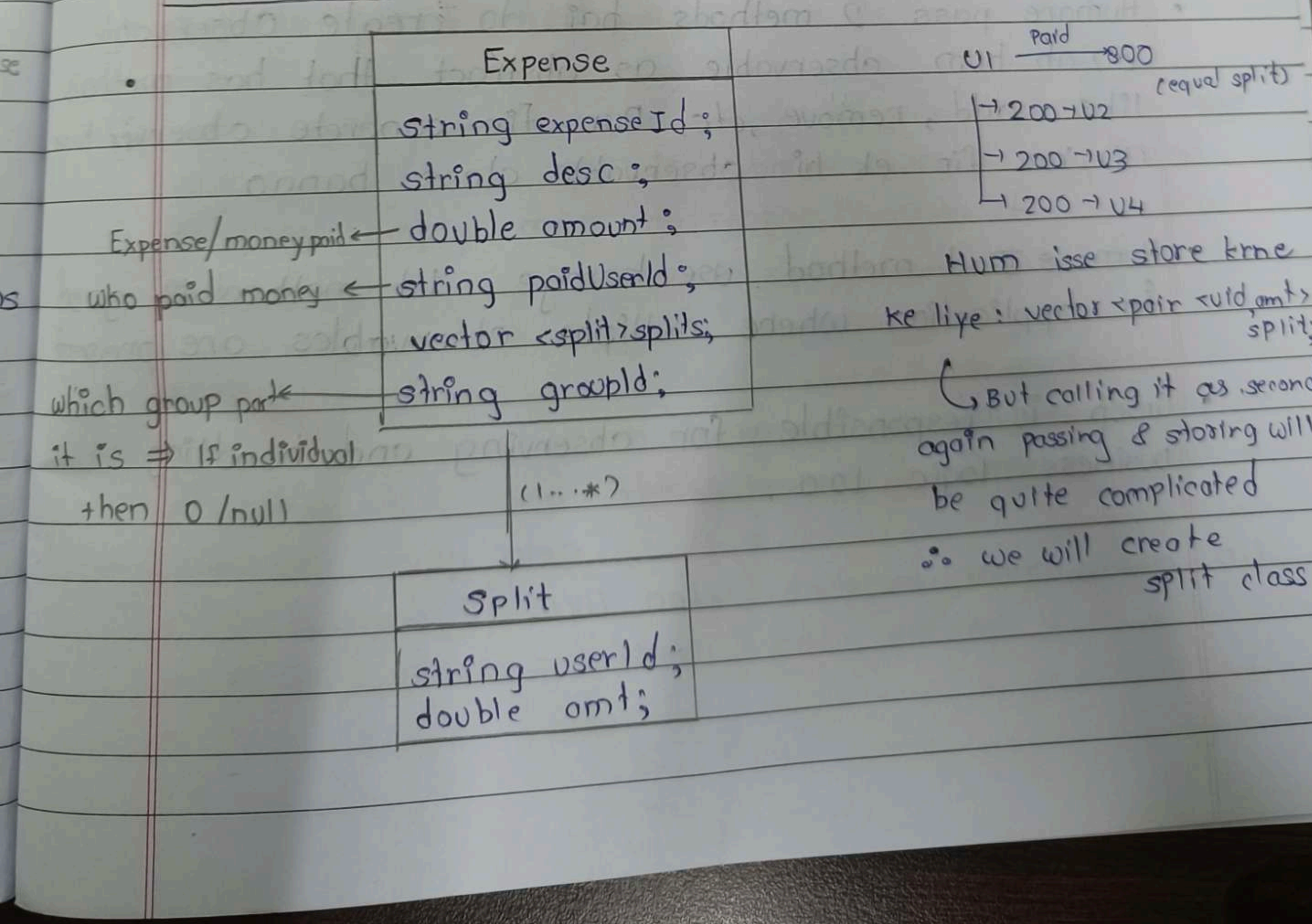
- We will use ids to add in group, split expense based on ids.
  ∴ In map <String, ...>
  ids

- How values is stored in map

id	value-they-own
U2 :	200 → U1 ko 200 dena
U3 :	300 → U1 ko 300 dena
U4 :	500 → U1 ko 500 dena



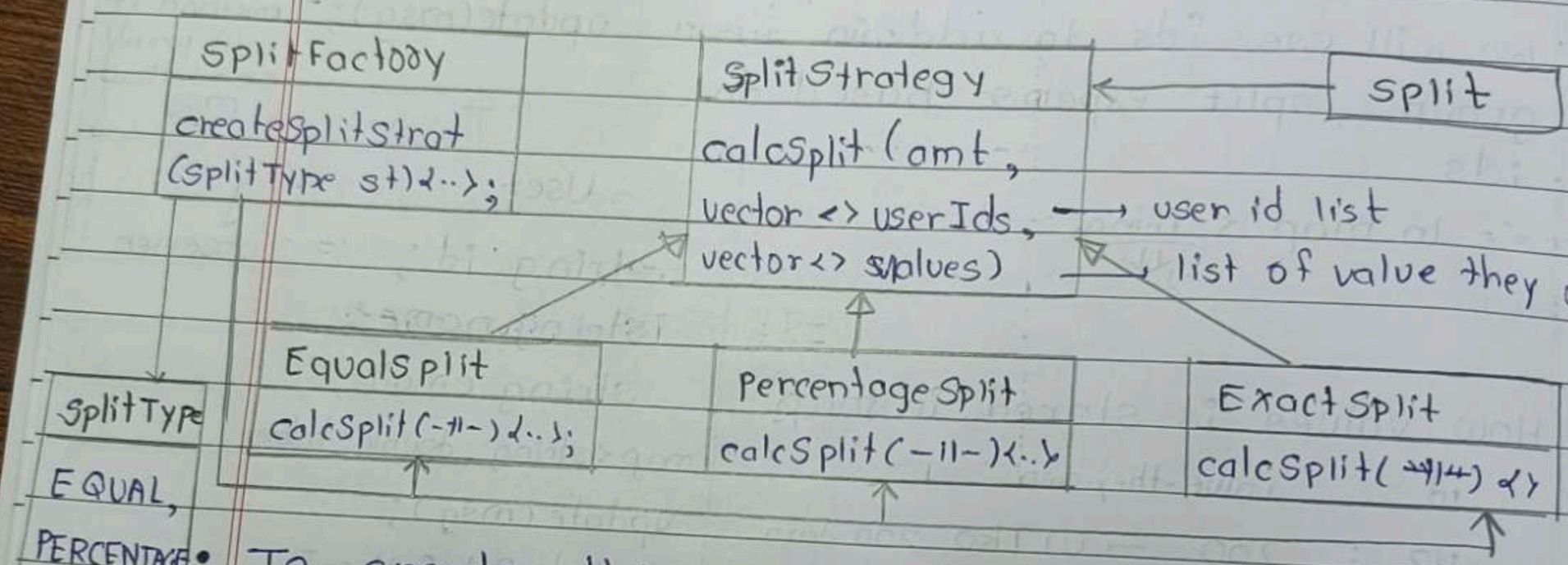
- updateBal() → will update value in our balance map



U1 —Paid→ 800
(equal split)
→ 200 → U2
→ 200 → U3
→ 200 → U4

Hum isse store krne ke liye: vector<pair<uid, amt>> split;
↳ But calling it as second, again passing & storing will be quite complicated
∴ we will create split class

- Now we need different strategies to split money among all the members and we will ~~create~~ use strategy pattern so that we can add more strategy in future.

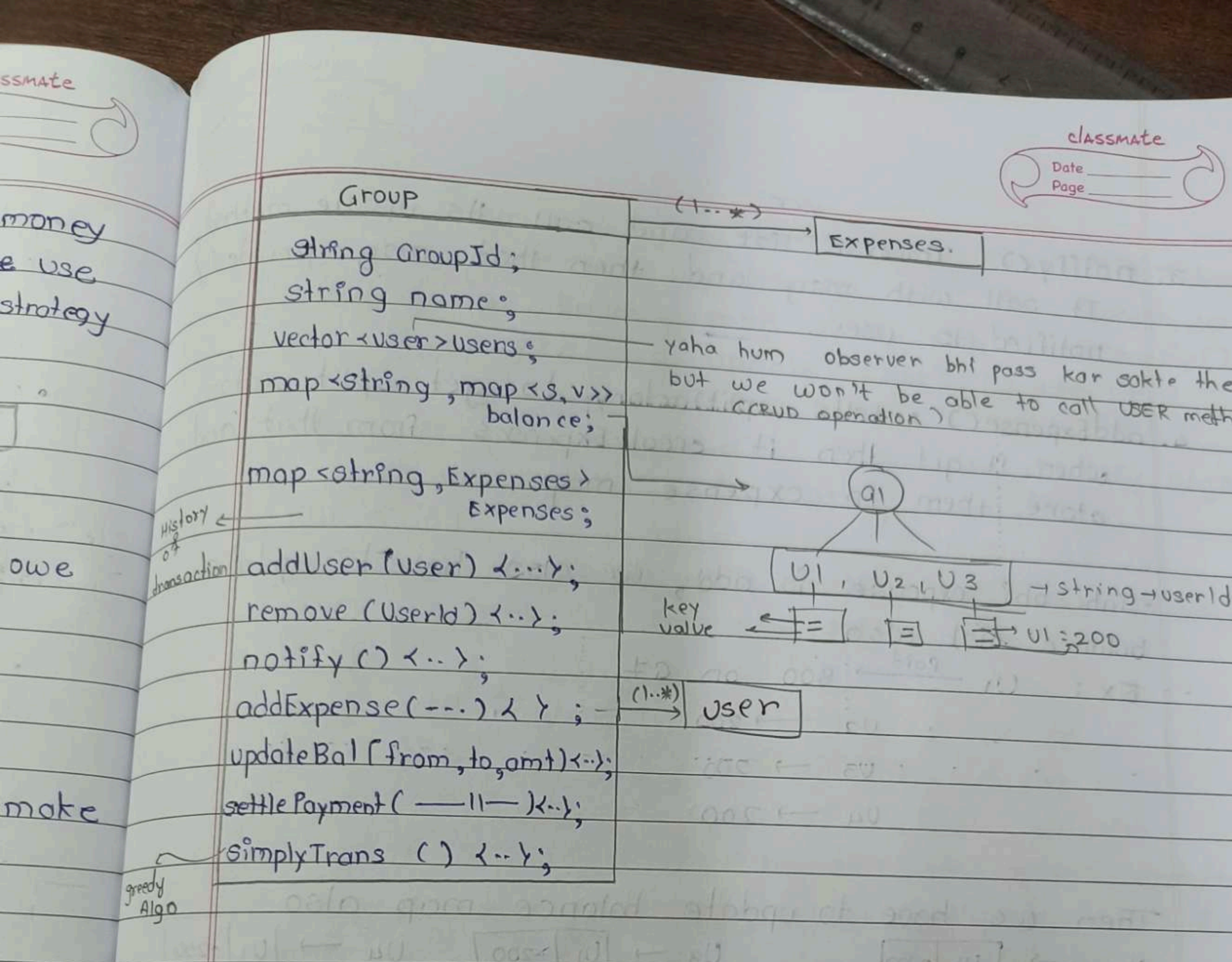


- To create this splitStrategy we will need to make splitFactory.

- Humare pass 2 methods hai to create Observable (i) ab hum observable as abstract that has methods like add, remove, etc. Aur fir concrete observer bana (ii) Yaa fir ek hi observable class banao

- Hum 8 IInd method use karege. 1st method is good in application where multiple observables are present

- Group is responsible for observing and handling business logic too.



- Details about methods

1. addUser() : adds new user in list of user and also add new user in balance map also and provide empty map corresponding to it

   userId ——> < >

2. remove() : remove user from users. Traverse in list → getUserbyId() and then remove it. Remove from balance map also.
   And remove() will only remove user when all expenses of that user were settled

user

3. notify() : Traverse list and call its update method. It call with msg and then that msg printed / notified to user.

4. addExpense() : tells splitFactory to get split vector when it get then it create Expenses from that and store them in expense map

- Jab bhi expense is add, we also have to update balance list also

Ex : U1 $\xrightarrow{\text{Paid}}$ 800 on a7

U2 → 200

U3 → 200

U4 → 200

Then we have to update balance map also

U2 → | U1 | -200 |

U3 → | U1 | -200 |

U4 → | U1 | -200 |

U1

U2	+200
U3	+200
U4	+200

↳ To implement this humare pass updateBal() hai

5. settlePayment() calls updateBal() and then tell it to update value in Balance map.

Ex : U2 paid 200 to U1

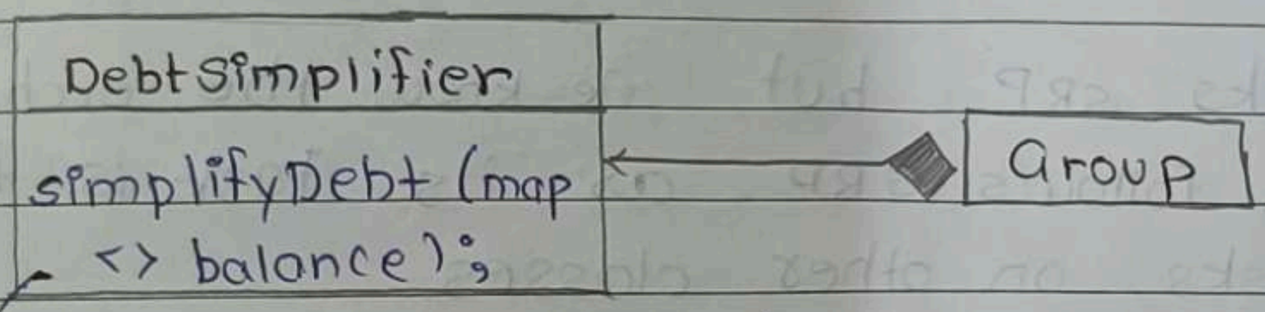
U2 → | U1 | 0 |

U1 → | U2 | 0 |

↳ agar hum addExpense() / settlePayment() call kare, inn dono case mein updateBal() call hoga which will ~~calls~~ update Balance map and also calls notify() jo saare user ko update / notify karta hai.

- Humare requirement mein transaction simplify algo ko implement karna hai using greedy Algo.

- To create simplifyTrans() method ek aur class banana padega - DebtSimplifier



↳ after this method executes it gives updated map of balance in group also & user too.

7. simplifyTrans(map<>balance) : calls balance map & then it simplify balance map. Then DebtSimplified returns updated and simplified map to simplifyTrans() LL> fir group ka balance map bhi update karte.

- Ab last class - Facade / Orchestrator class jo interact karega humare saare complex system se.

Splitwise
map<string, user> Users;
map<string, expenses> Expenses;
map<string, group> groups;
USER (CRUD opes);
GOUPS (CRUP ops);
//User mgmt

- We have completed all requirement except one
  ↳ Add individual expense
  To achieve this humne already updateBal() method banaya tha in User class, to successfully implement it humare pass 2 options hai -

  i) create userManager class to handle it
  ii) Yaa humare orchestrator class ko ~~b~~ list of user diya hi hai toh usse hi bolo to manage.
  user (✓) using this

  ↳ it breaks SRP but we know the orchestrator class do not follows SRP. as its task is to delegate tasks on other classes.

Debt simplification Algorithm - details

Let's take example to understand this

- Aditya
- Rohit
- Manish
- Saurav

① Aditya $\xrightarrow{\text{Paid}}$ 800 (equal split for lunch)

- Rohit : 200
- Manish : 200
- Saurav : 200

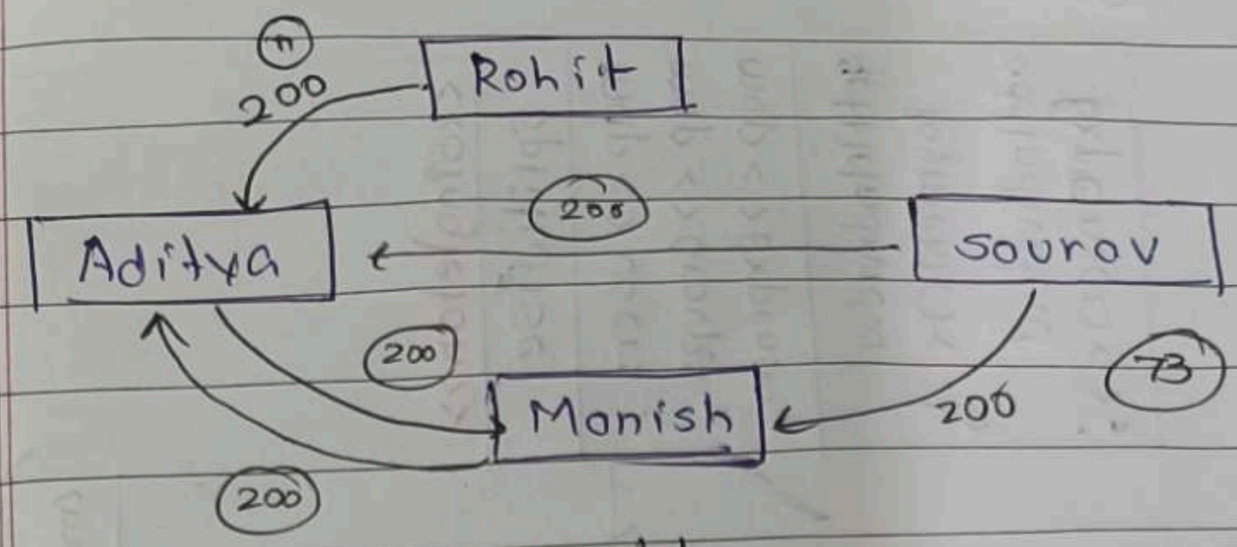
} to Aditya

② Manish $\xrightarrow{\text{paid}}$ 700 (exact split)

- Saurav : 200
- Aditya : 200

Manish : 300

Understanding through diagram



+200 } settle no need to pay to each other
-200 }

Ab total transaction 3 hue due to greedy ab hum usse aur reduce karege

↓↓

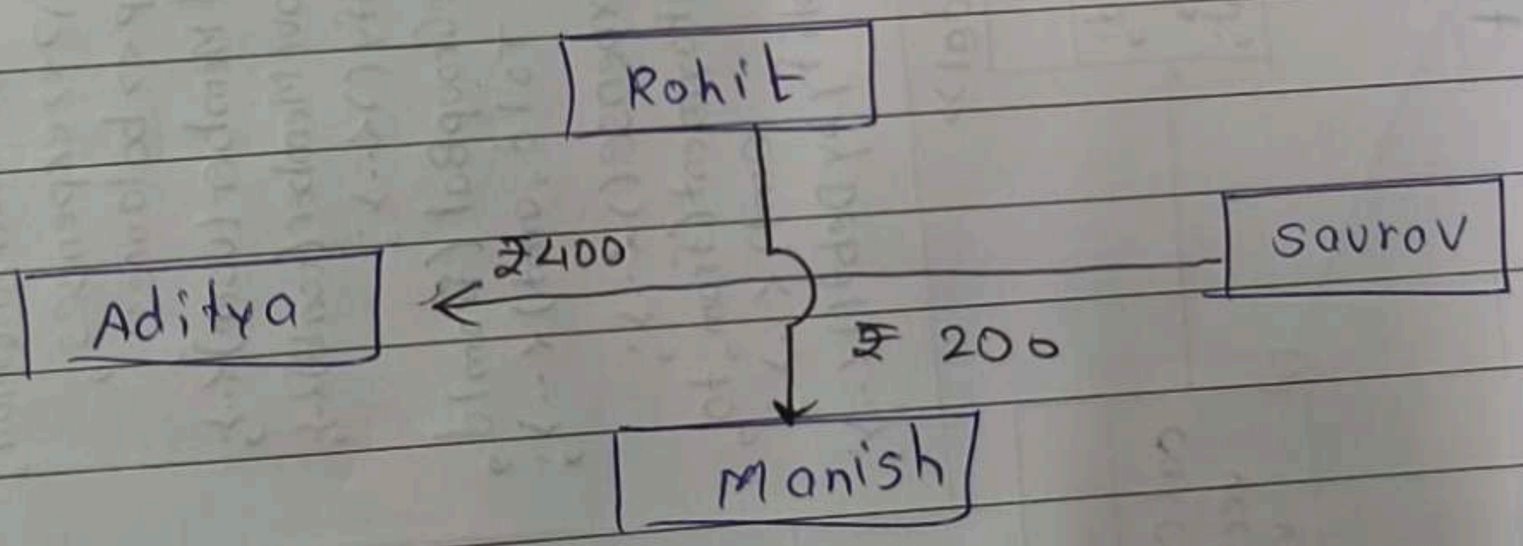
To do this

Net Gain / Loss

- Net Gain and Net Loss

→ Aditya (+400)	Sourav ⟶ (-400) } debit
~~debit~~ → Manish (+200) credit	Rohit ⟶ (-200) }

Ab humne divid kar diya debit & credit mein



T3 ⟹ T2

↳ This is simplify transaction

Final UML Diagram