AGENDA

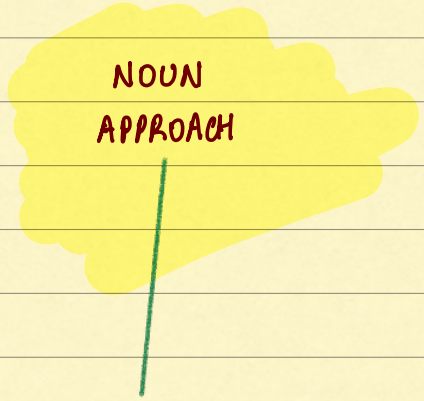
1.> class diagram

2.> schema design

3.> Take input via cmd line

start by 9:08 PM IST

*) CLASS DIAGRAM:



NOUNS for which we store data.

- [ ]
- [ ]
- [ ]
- [ ]

- [x] USER [Attributes]
- [x] Expense
- [x] Group
- [ ] Transaction

TRANSACTION:

A ——— B

E_1

E_2

E_3

E_4

A ——— C

E_5

E_6

Settle up ✓

(T_1) -> A Needs to pay B 500

(T_2) -> A Needs to take from C 200

TRANS.

(T_1) ⟶ A Needs to pay B 500

A Needs to take from C 200

~~☐~~ TODAY

☐.

↓

(A)

A paid 300 to B ✗

TRANSACTION

T_1

EXP.S

E_1

E_2

E_3

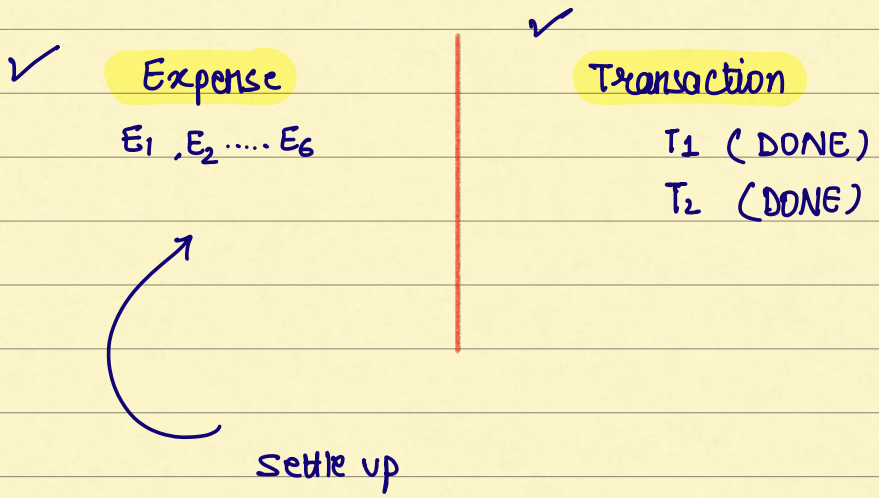
E_4

E_5

E_6

*) WHAT HAPPENS WHEN USER ✓ ON TRANSACTION:

case 1:> Move all DONE transactions on Trans. class



HOW MUCH MONEY TO PAY ?

USER, GROUP, EXPENSE:
↑↑

Case 2:

✓ A needs to pay B 500
✓ A needs to take from C 200

EXPENSE
A —500→ B

DUMMY EXPENSE
A —Paid→ 500

Expense
Paid: A:500
HADTOPay: [A:0, B:500]

E_1
E_2
E_3
E_4
E_5

B: +500
A: -500

(A) A owes 500/- B (T_1)

E_6: A Paid 500.
A hadTOPay = 0]

Paid [A:500]
hadTOPay [A:0, B:500]

B hadToPay = 500 ↓ (owes)

A == B == 0

DUMMY EXPENSE

	(Y)	Y:S
LUNCH (1)	1000	500/-

S owes Y = 500

Expense → Paid

saurav → 500

owed: 0

yash: 500

*) CLASSES:

① USER

- id
- Name
- PhoneNo
- Password

② Expense

- id
- description
- Amount
- PaidBy <user, Int> ✗
- hadToPay <user, Int?> ✗
- createdAt
- ExpenseType

③ Group

- id
- name
- Admin
- List<expenses>
- List<user>

④ ExpenseType

DUMMY, REAL

" some expenses can be part of Group, some cannot"

Expense <> Group (Relation)

1.> Group {
<Expense>

}

2.> Expense {
Group

}

GroupMapping {
Group
Expense
}

*] HADTOPAY | PAIDBY :

.> Attribute of Relation b/w: USER : Exp.

PaidBy <USER, AMOUNT>

User can pay - currency

A1:

ExpensePaidBy
user
expense
Amount

ExpenseHadToPay
user
expense
Amount

#A2:

ExpenseUser
user
expense
amount
Type

ExpenseUserType
PAID-BY
HAD_TO_PAY

① USER

- id ✓
- Name ✓
- PhoneNO ✓
- Password ✓

② Expense

- id ✓
- description ✓
- Amount ✓
- ~~PaidBy <user, Int>~~ ✗
- ~~hadToPay <user, Int>~~ ✗
- createdAt ✓
- ExpenseType •
- createdBy •

③ Group

- ✓ id
- ✓ name
- • Admin
- • List<expenses> •
- • List<user>

G:User
M:M

④ ExpenseType

DUMMY, REAL

G:E M:1

⑤ ExpenseUser

- • user
- • expense
- ✓ amount
- • Type
- ✓ id

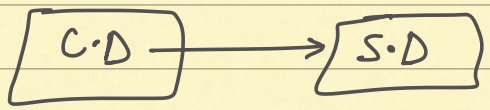
⑥ ExpenseUserType

PAID-BY
HAD_TO_PAY

Expu: U (M:1)
Exp: E (M:1)
Expu: type (M:1)

10:24 PM 1st

*) SCHEMA DESIGN:



S1.) Convert all classes → tables

users, expenses, groups, expense-user-type, expense-user, expense_type

S2.) PRIMITIVE ATTRIBUTES

1.) users

id, name, phone_no, passwd

2.) expenses

id, description, amount, created-at

3.> group
id, name

4.> expense-users
id, amount

5.> expense-user-type
id value

6.> expense_type
id value

S3.> NON-PRIMITIVES:

1.> users

id, name, phone_no, passwd

2.> expenses

id, description, amount, created_at,
created_by_id, expense_type_id, group_id

3.> group

id, name, admin_user_id

4.> expense_users

id, amount, exp_user_type_id, user_id, exp_id

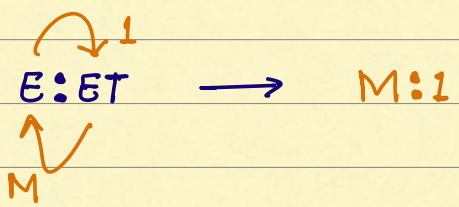
5.> expense_user_type

id value

6.> expense_type

id value

Expense:



E: user (created) ⟶ M:1

SCHEMA:

1.) **users**

id, name, phone_no, passwd

2.) **expenses**

id, description, amount, created_at,
created_by_id, expense_type_id, group_id

3.) **group**

id, name, admin_user_id

4.) **expense_users**

id, amount, exp_user_type_id, user_id, exp_id

5.) **expense_user_type**

id value

6.) **expense_type**

id value

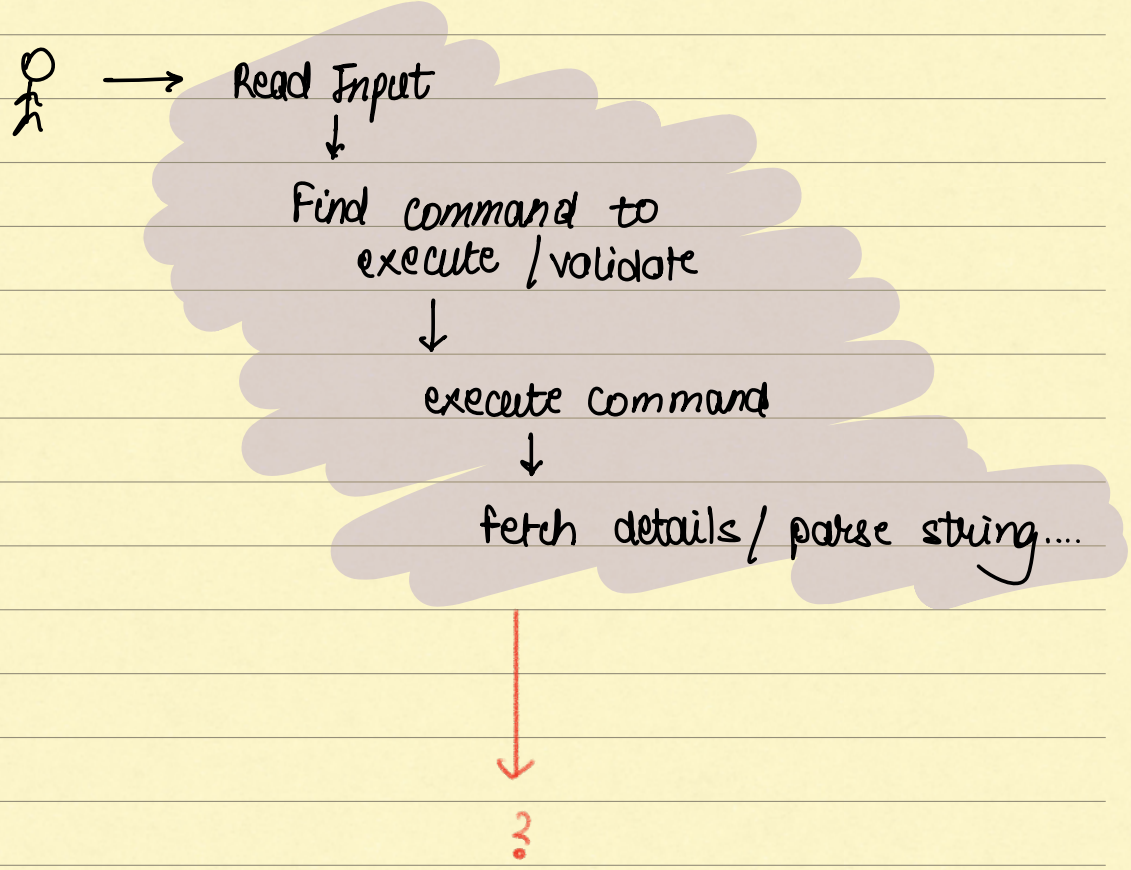
7.) **group_users**

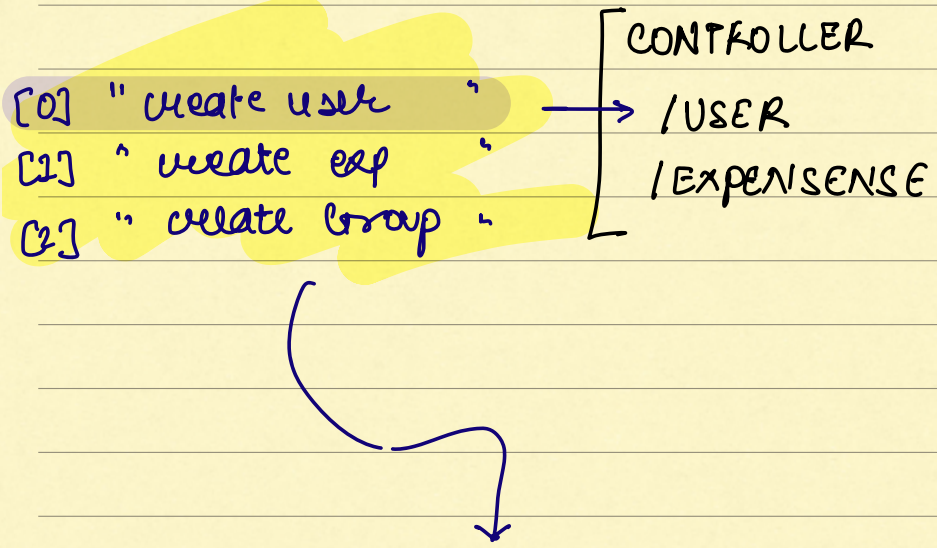
id, group_id, user_id

*) TAKING INPUT VIA CMD LINE:

"create-user yash _ _ _"
"create-expense "descp" _ _ _"
⋮

client/main





Main / client ...

AI:

In Main Method:

```
Main() {

while(true) {

String input =    // take input

List<String> inputwords = input.split()

→ if (inputword.get(0).equals("create user")
```

→ createUser...()

}

→ if (inputWord.get(0).equals("create map")

→ createUser...()

}

:

.

.

.

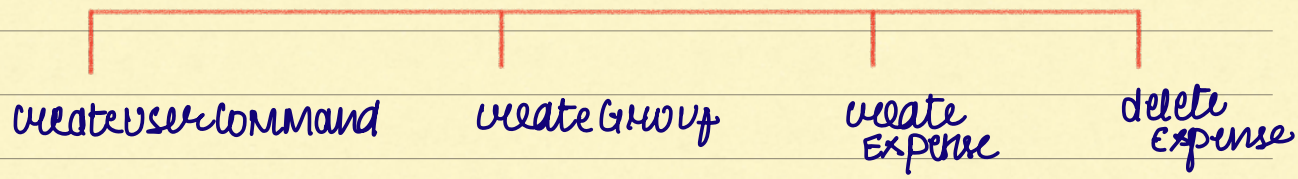
}

A2: Better Idea : REGISTRY:

1.) Whether string matches with command OR NOT
2.) execute the command

<< Command >>

boolean matches(string input)
void execute(string input)



In main:

```
main(){
    List<Command> cc= [.....];

    while(true){
        string input =
        ---- spInput = input.split()

        for(Command c: cc){
```

```
if ( c.matches(input)
    c.execute();
    break;
```

```
}
}
}
```

Approach 3:

```
CommandRegistry{

List<Command> list = [.....];

addCommand() {

}

removeCommand() {
}

execute( Input string ){

for ..... () {
```

HOMEWORK:

1.) create Models
2.) use annotations ...
3.) implement Command pattern
4.) → "User can Register Card to Card"