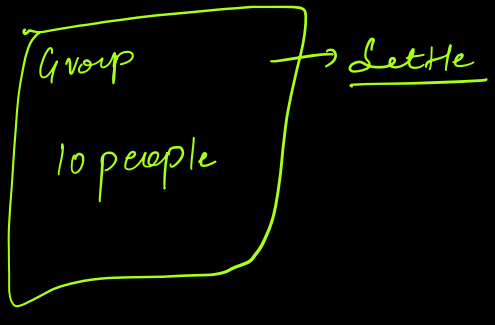
Today's Agenda :-

1) Algo to find min no of txn required for settle up.
2) Class Diagram
3) Schema Design
4) Taking Inputs via Command line

Command Design Pattern.
↳ Behavioural.

5 expense



#lended = #borrowed

N no of people

N log N

Algo:- Divide the users into 2 buckets ✓

Lenders — who've paid extra

Borrowers — who've paid lesser.

{ A : +1000 } { E : +750 }

{ B : −50, C : −1250, D : −450 }

max from bucket 1

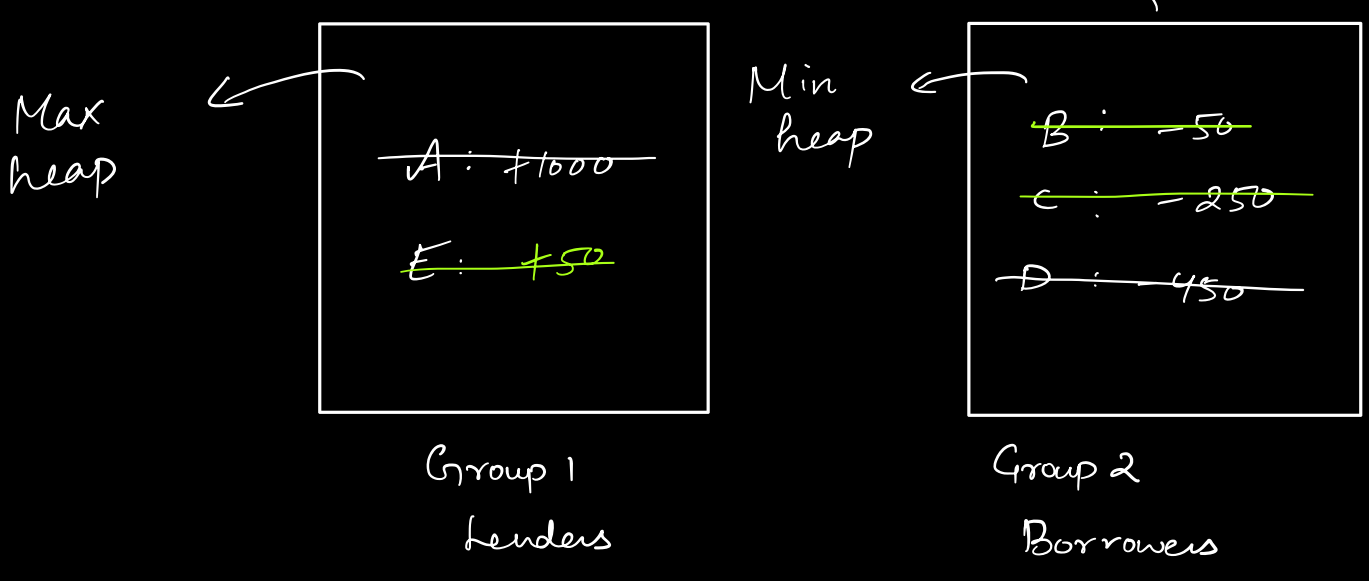
min from bucket 2

① C → A : 1000

② D → E : 450 ✓

③ C → E : 250

④ B → E : 50



Borrowers should pay to the lenders.

I want to list down the txn required for the settle up

Entities :- Nouns about which you want to store data in your database.

Go through req gathering

User ✓

Expense ✓

Group ✓

Transaction

User
id name email password

Expense
id description amount group Map<User, amount> lenders; Map<User, amount borrowers; ExpenseType

who paid how much

who borrowed how much

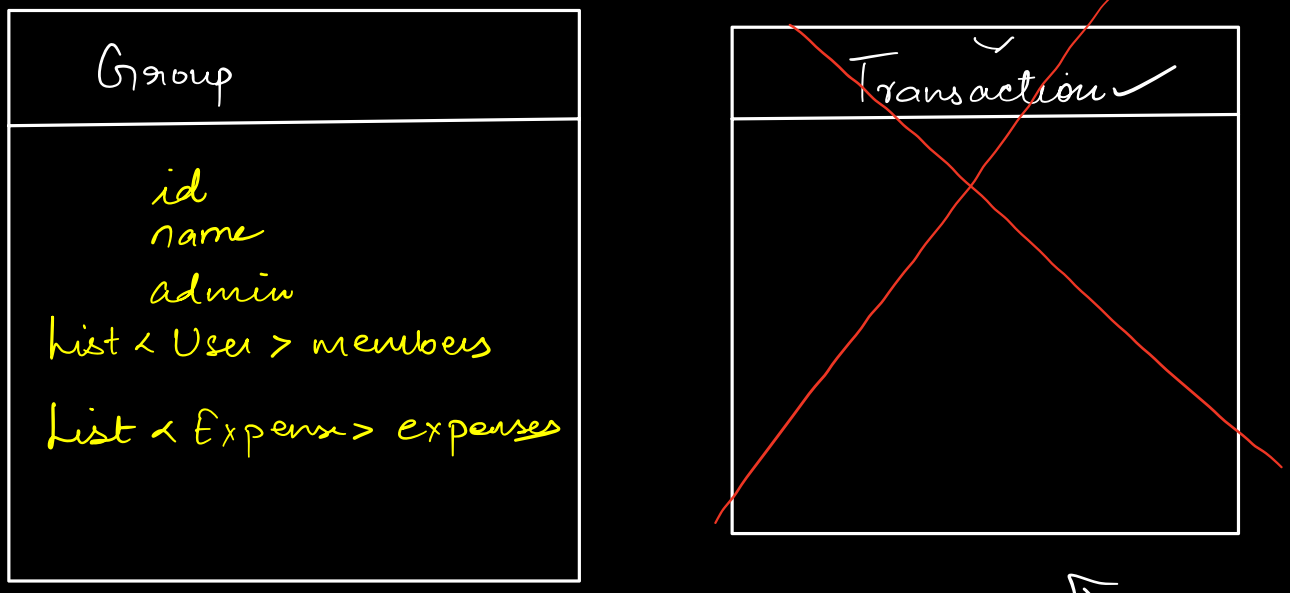
₹ 1000

A	200	A	250
B	800	B	250
		C	250
		D	250

lenders :- { (A, 200), (B, 800) }

borrowers :- { (A, 250), (B, 250), (C, 250), D(250) }.

ExpenseType
DUMMY, ACTUAL



Example

- Expense on a dinner for ₹ 3000
- lenders A : 2000 B : 1000
- borrowers A: 700 B: 800 C: 1000 D: 500

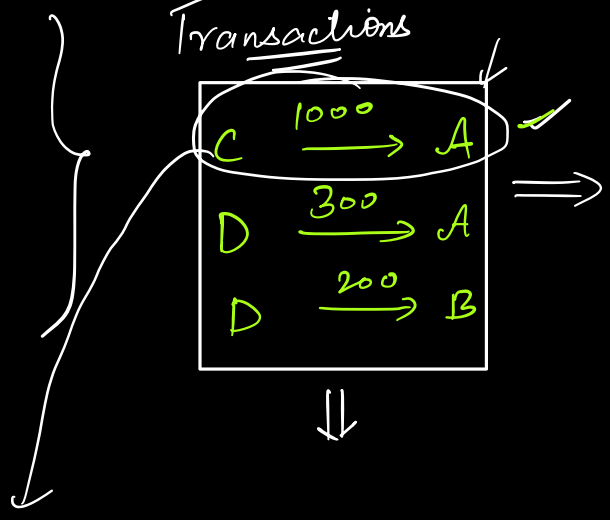
Settle up ⇒ list < Txn > required so that everyone's balance becomes zero.

A : +300

B : +200

C : -1000

D : -200



Doesn't make sense to persist this data in the db.

When a user marks the transactions as done | completed, we will have to mark the effect of the txn on the amount.

↳ Persist somehow the executed txn

⇓

Create a dummy expense.

C $\xrightarrow{1000}$ A

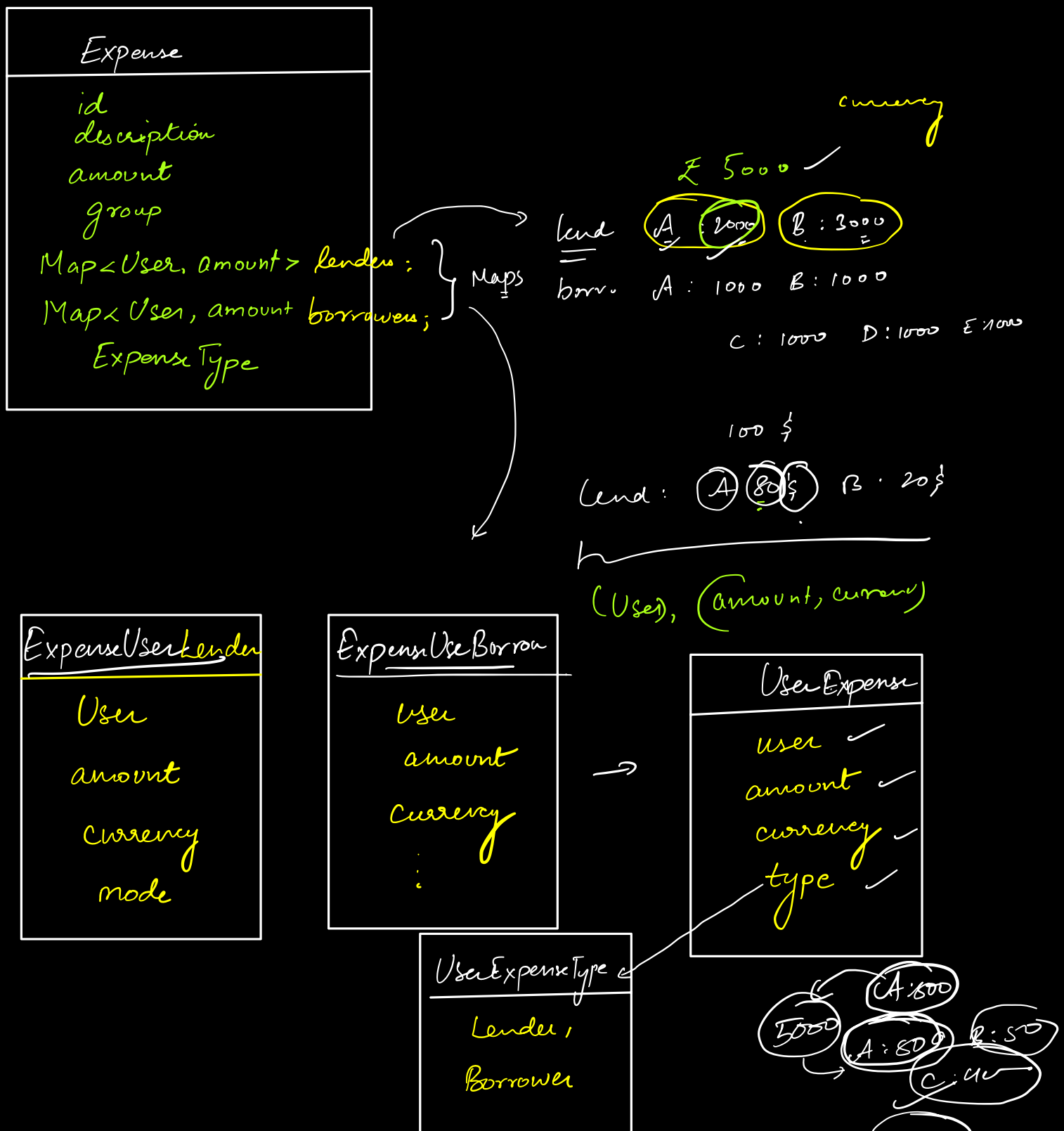
amount : 1000
lender : C : 1000
borrower : A : 1000

A : +1750 − 450 = +1300
B : −50
C : −1250
D : −450 + 450 = 0

C $\xrightarrow{1250}$ A
D $\xrightarrow{450}$ A
B $\xrightarrow{50}$ A

dummy expense
amount : 450 ✓
lender : D : 450
borrower : A : 450

Conclusion :- Instead of creating a separate class, we'll create a dummy expense which have the same impact as of a executed / completed txn.


Expense
id
description
amount
group
Map<User, amount> lenders;
Map<User, amount> borrowers;
ExpenseType
Maps
currency
£ 5000
lend
A : 2000
B : 3000
borr. A : 1000 B : 1000
C : 1000 D : 1000 E : 1000
100 $
lend : A 80$ B : 20$
(User), (amount, currency)
ExpenseUserLender
User
amount
Currency
mode
ExpenseUseBorrow
User
amount
Currency
UserExpense
user
amount
currency
type
UserExpenseType
Lender,
Borrower
5000
A:800
A:800
B:50
C:40

Expense
id description amount group list < UserExpense > ExpenseType

5000 E

Lend A : 1000 B : 4000

borr A : 2500 B : 2000 C : 500

{ {user : A, amount : 1000, currency : INR, type : lender},

{ — B, 4000, INR, lender },

Expense — UserExpense : 1 : m

1 M

1 1

Group — Expenses : 1 m / 1 1

M : M { Group : 1 m / m 1 — Members

User
id name email password

ExpenseType
DUMMY, ACTUAL

Group
id name admin list < User > members List < Expense > expenses

Expense
id description amount group list < UserExpense > ExpenseType AddedBy

UserExpenseType
Lender, Borrower

UserExpense
user amount currency type

Schema Design

group = admin (User)

M:1 { 1 = 1, n = 1 }

M:1 { Expense: 1, M — Group: 1, 1 }

users

id	name	email	password

expenses

id	desc	amount	group_id	expense_type_id	

groups

user_id

id	name	admin_id	

user_expenses

id	user_id	amount	currency	expense_id	user_expense_type_id

expense_type

id	value

user_expense_type

id	value

groups_users

group_id	user_id

Take inputs via Command line :-

↳ Behaviour

Register ___
_ updateProfile _

input → Read input → find the command from the input → Parse the inputs params

!

Call the correct code for the command.

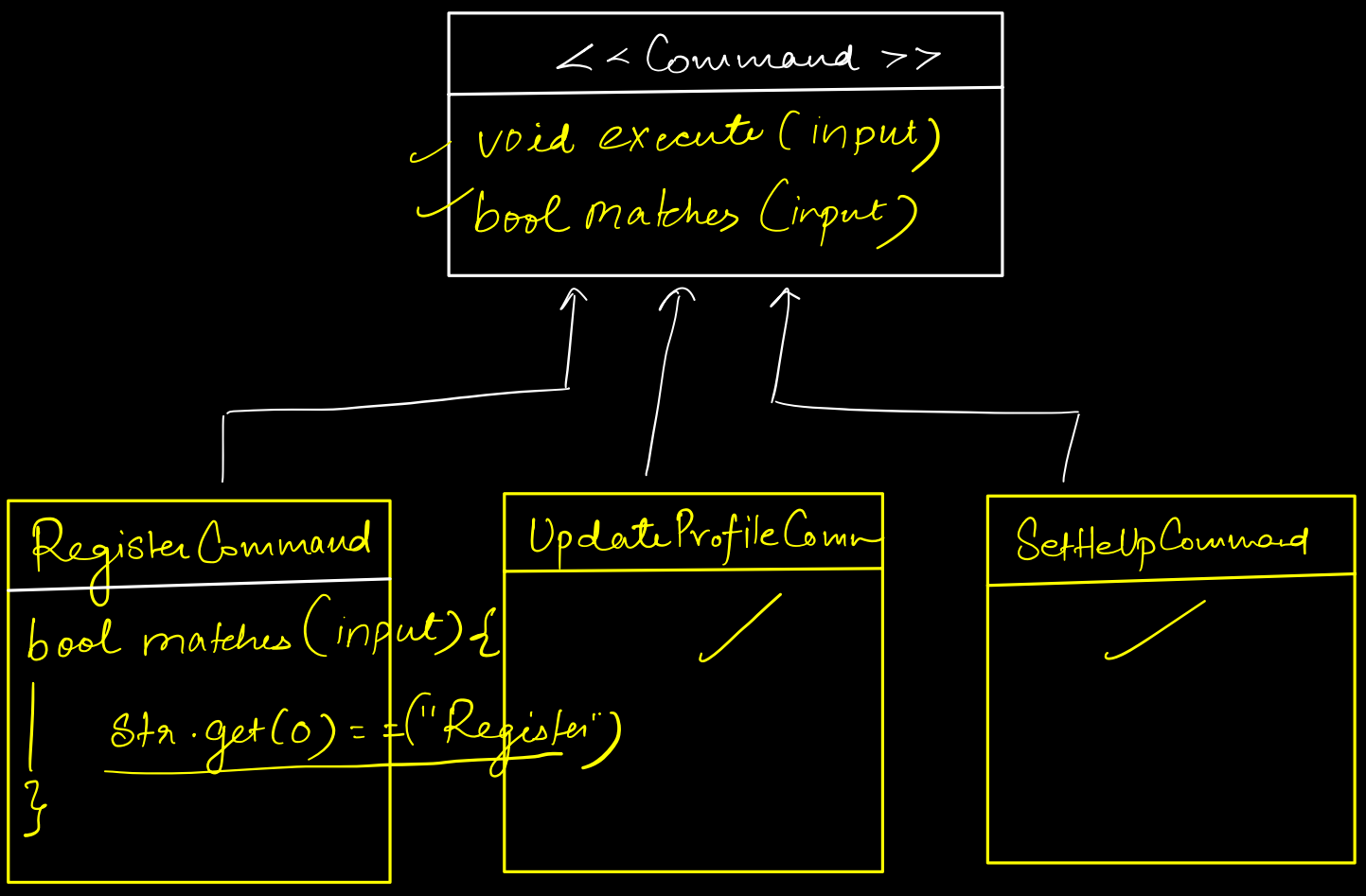
Register using OOP style

```
String input = scanner.nextLine();
List<String> words = input.split(" ");
if (words.get(0).equals("Register")) {
    //
} else if (words.get(1).equals("UpdateProfile")) {
    //
}
else if (                    ) .
```

Too many if else (SRP & OCP violation)

We need better way to check if a String is a command

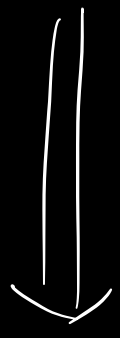
We need to execute the command.



```
List<Command> commands = {...}

input = scanner.next();

for (Command cmd : commands) {
    if (cmd.matches(input)) {
        cmd.execute();
    } break;
}
```



```
input = scanner.next();

CommandExecutor.execute(input)
```

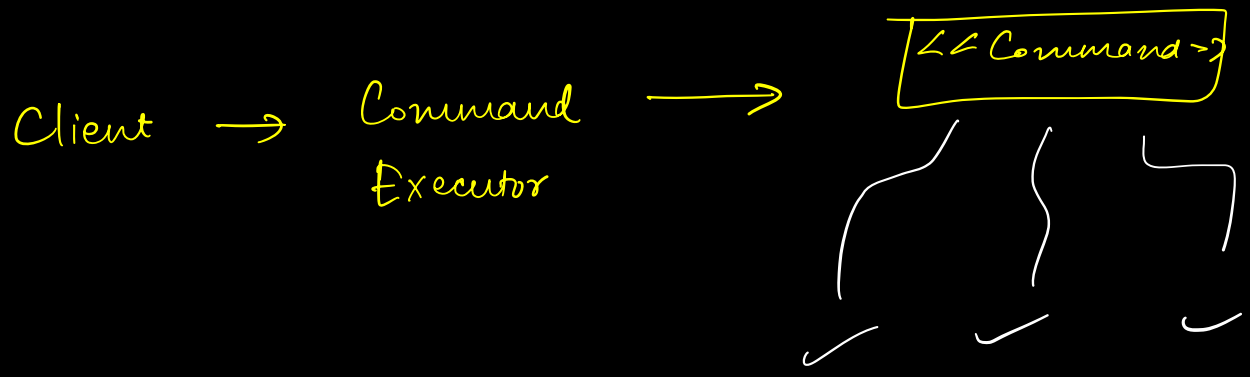
CommandExecutor

```
List<Command> commands = {...}

void addCommand(Command cmd) {
    commands.add(Command)
}

void execute(String input) {
    for (Command cmd : commands) {
        if (cmd.matches(input)) {
            cmd.execute();
        } break;
    }
}
```

static



Command Design Pattern

1

User Expense

Thank You