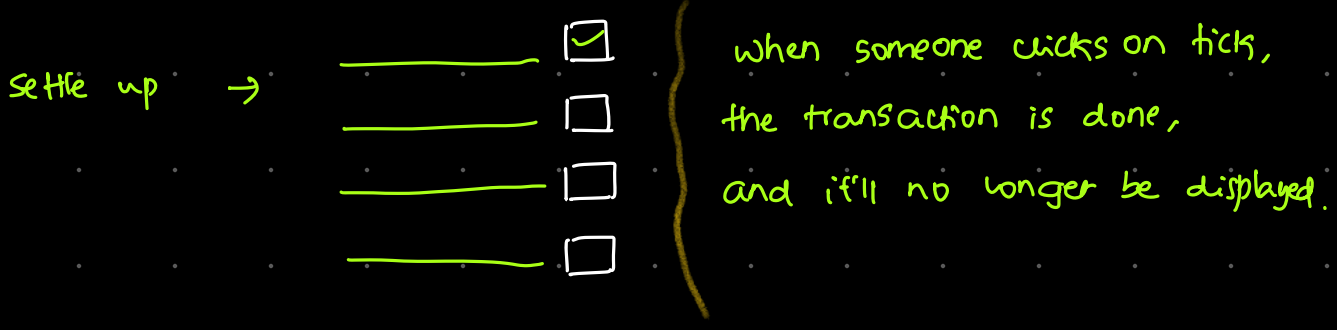
Requirements

- Users can register and update their profiles.
- A user's profile should contain at least their name, phone number and password
- Users can participate in expenses with other users
- Users can participate in groups.
- To add an expense, a user must specify either the group, or the other users involved in the expense, along with who paid what and who owes what. They must also specify a description for the expense.
- A user can see their total owed amount
- A user can see a history of the expenses they're involved in
- A user can see a history of the expenses made in a group that they're participating in
- Users shouldn't be able to query about groups they are not a member of
- Only the user who has created a group can add/remove members to the group
- Users can request a settle-up. The application should show a list of transactions, **which when executed** will ensure that the user no longer owns or receives money from any other user. Note that this need not settle-up with any other users.
- Users can request a settle-up for any group they're participating in. The application should show a list of transactions, which if executed, will ensure that everyone participating in the group is settled up (owes a net of 0 Rs). Note that it will only deal with the expenses made inside that group. Expenses outside the group need not be settled.
- When settling a group, we should try to minimize the number of transactions that the group members should make to settle up.

Note: All tests will be performed in one go. The application doesn't need to persist data between runs.

Nouns about which we store information.

1. user 2. expense 3. group 4. transactions.
   ↓
   Calculated at runtime.
   using expenses.



When someone clicks on 'tick'.

Option1: Create a class called 'DoneTransactions', and add the transaction to this.

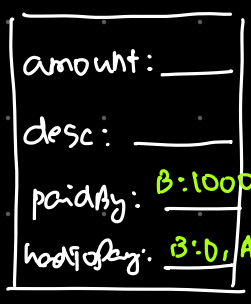
For a group, you will calculate list of transactions, filter the done transactions.

Not so great!--

Option2: When a transaction is done, you try to add a dummy expense.

Such that when we calculate list of transactions next time, we don't get the done transaction.

Ex:

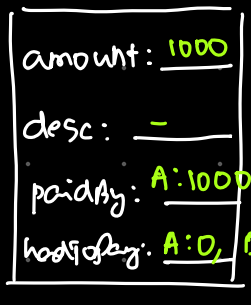


expenses.

Transactions.

A -> B : 1000 ☑ {B has paid 1k extra}

We add a dummy expense.



Before transaction:

Person	extra_amount_paid.
A	0 - 1000 = -1000
B	1000 - 0 = 1000

A -> B : 1000 ☑

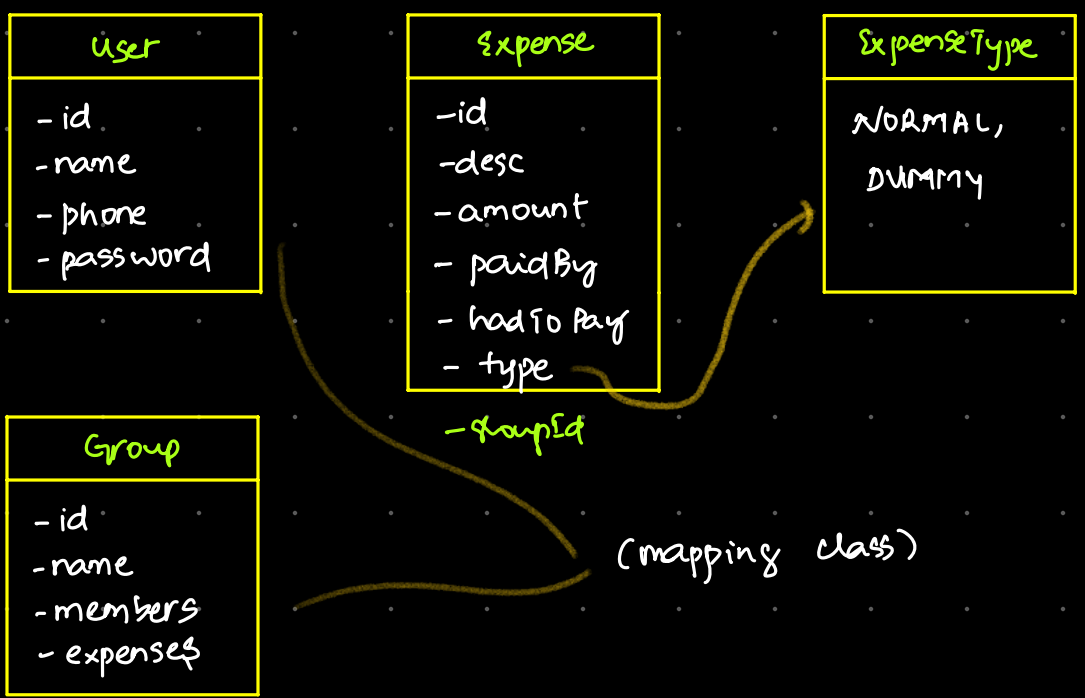
After transaction

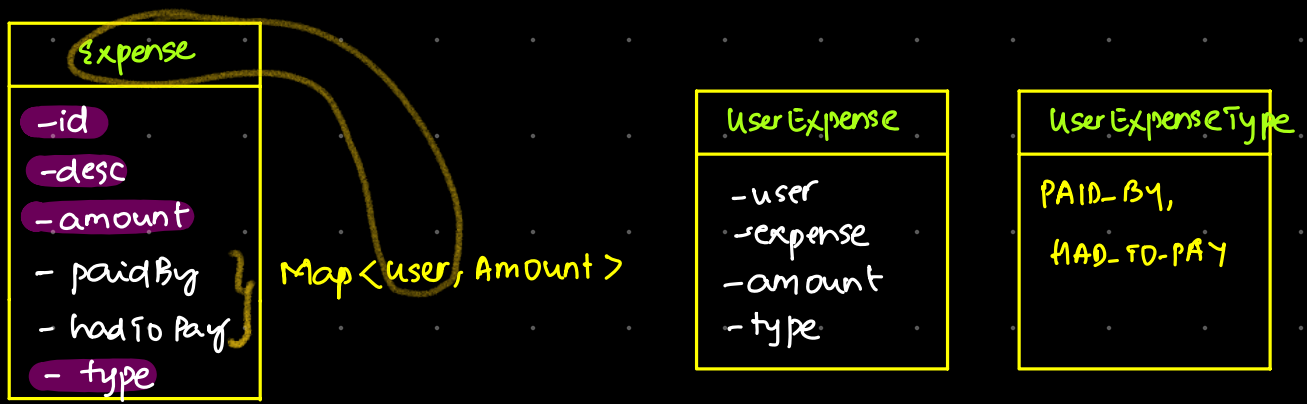
Person	extra_amount_paid.
A	0-1000 +1000-0 = 0.
B	1000-0 +0-1000 = 0.

NO transactions, everyone is settled up.

class diagram

1. user
2. group
3. expense





schema diagram → HW.

---

u1 UpdateProfile robinchwan

u1 is updating their profile password to "robinchwan"

---

u1 AddGroup Roommates

u1 is creating a group titled "Roommates"

---

u1 AddMember g1 u2

u1 is adding u2 as a member of the group "Roommates" (which has the id g1)

---

u1 MyTotal

u1 is asking to see the total amount they owe/recieve after everything is settled.

---

u1 History

u1 is asking to see their history of transactions (whether added by themselves or someone else)

---

u1 Groups

u1 is asking to see the groups that they're a member of

---

based on the commands, you should be able to call the corresponding methods.

Easiest way

```
main()
    while(true)
        String input = Read i/p command line by line.
        List<String> inputWords = input.split(" ");
        if(inputWords.get(1).equals("register"))
            // read username, password
            // call methods accordingly
        if( _____ . _____ )

        if( _____ , _____ )
```

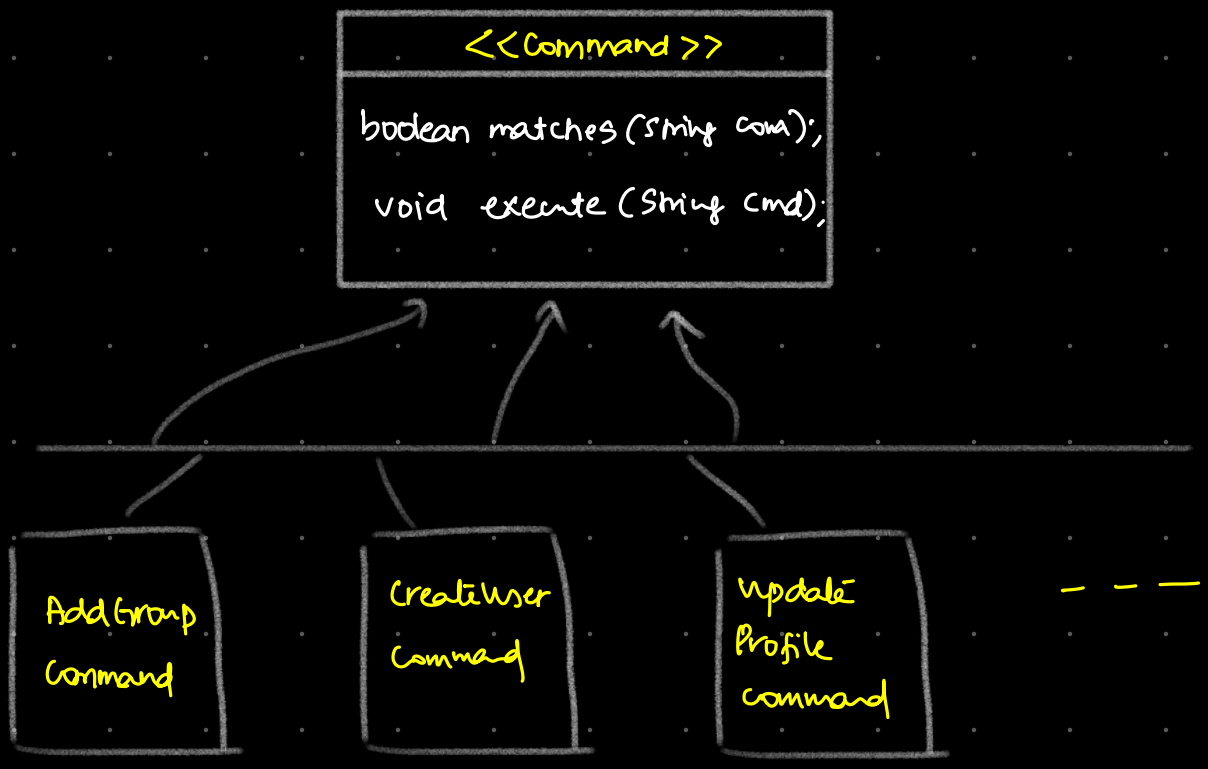
Breaking SRP.

For every command

① matches with the string (valid or not)

② execute the given string

<<Command>>




```
main()
    List<Command> commands; // all commands
    while(true)
        String input = Read i/p command line by line.
        for(Command command : commands)
            if(command.matches(input))
                command.execute(input)
```

New commands keep coming in.

Executor

CommandRegistry

```
List<Command> commands;

addCommand(Comand c)
    commands.add(c)
removeComnad(Comand c)
    commands.remove(c)
execute(String input)
    for(Command c : commands)
        if(c.matches(input))
            c.execute(input)
```

```
main()
    CommandExecutor cexecutor;
    while(true)
        String input = Read i/p command line by line.
        cexecutor.execute(input);
```

Comnad execution is delegated to comnd executor.