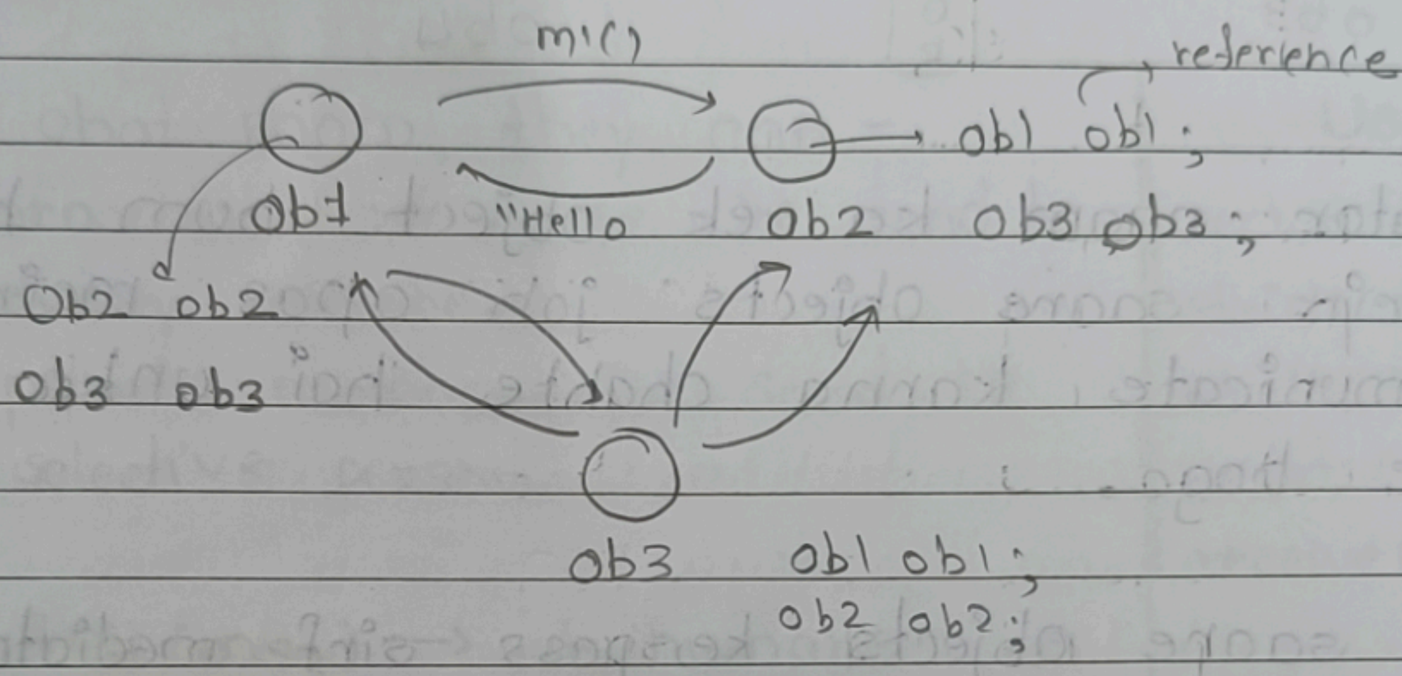
# Lecture 35 : Mediator Design Pattern

## # Introduction

- simple and widely used pattern

- suppose there are two objects and they need to communicate, so to communicate / interact they both need reference of each other.



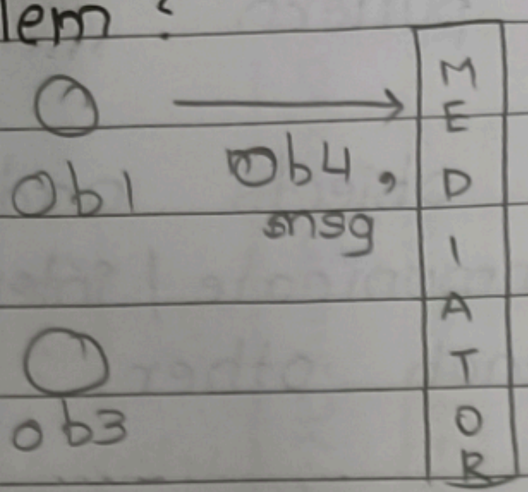

- Agar humne ob3 add kiya toh uska bhi reference ob2 and ob1 mein store karna padega and agar hum aise hi continue new objects banaye to communicate toh aise hi saare reference store karne padege.

- Total n objects hai toh (n-1) interactions lagege and utne hi reference store karne padege whether in list / vector.

- And this makes it tightly coupled and breaks OCP principle too for every object.

- And here comes entry of mediator design pattern.

# How mediator design pattern solves problem ?




- Mediator naam ka ek object hum banayege jismein saare objects joh apas mein communicate karna chahte hai unka reference store hoga.

- And saare objects ke pass sirf mediator ka reference hoga.

- suppose ob1 ko message send karna hai toh simple - send (ob4, msg);
  - ob4 → whom to send
  - msg → what to send.

- Inshort, mediator performs communication btw two objects without objects having reference of each other.

- Problem it is solving -
  1. loosely - coupled
  2. Not too much reference ing every object

# ChatRoom Example - Without Mediator Pattern

1. Multiple user interacts with each other.

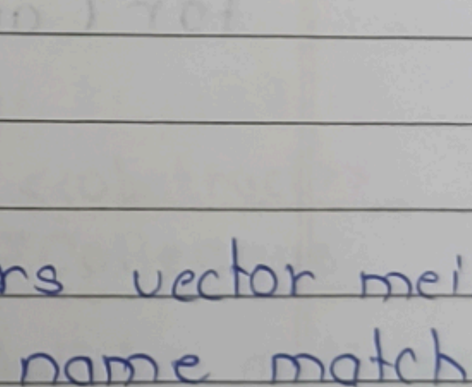
2. Now we have list of user of all users in our class

3. In chat room, they can -
   a) broadcast message i.e. send to every member
   b) private member i.e. send to selective person

4. sendAll (msg) → yeh humare users ke vector mein jaayega aur sabka receive() method call kar dega.

5. send (msg, to) → yeh humare users vector mein jayega check karega kiska ~~name~~ name match hota hai with to - name se and uska ~~notify~~ receive() method call kar dega.

6. User class will not be singleton class as multiple user will exist. Toh hum jithe objects banayege user class ke uthi users vector ka size badega exponentially as har new created obj bhi saare objs ke reference rkhega. i.e. (n-1) objects and too much memory taken.



7. And still it is manageable but what if we created complex ~~objects~~ methods to perform suppose -

mute() - To implement this humme vector<users> muted banana padega which will store users who are muted and if koi user joh uss list mein present hoga woh msg kar raha hai toh our task is to not notify abt that msg.

So now to sendAll() method mein hummein check lagana padega.

i.e. snippet of code

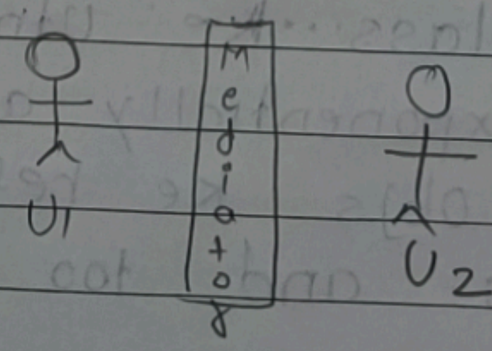
muted = {Mohan}

Mohan sends msg

```
for (auto user : users) {
    if (user -> getMuted (name))
        continue;
    else
        user -> receive';
```

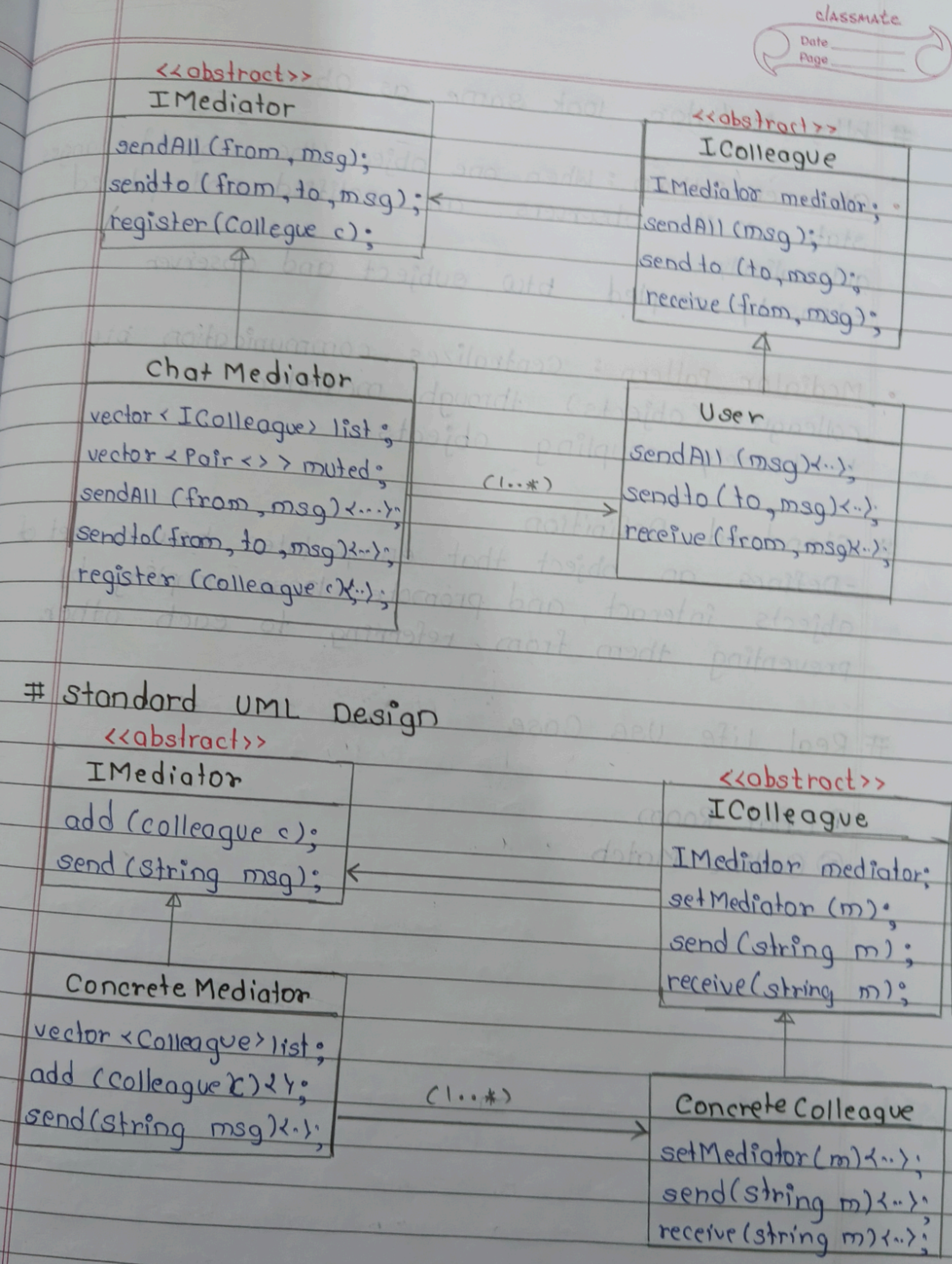
↳ And solution of this is mediator pattern

# Solution of Chat Room Problem : Mediator Pattern




user will be called collegue as they don't have reference of each other but communicates

∴ This makes users loosely coupled.


<<abstract>>
IMediator
sendAll (from, msg);
sendto (from, to, msg);
register (Collegue c);
<<abstract>>
IColleague
IMediator mediator;
sendAll (msg);
send to (to, msg);
receive (from, msg);
Chat Mediator
vector < IColleague> list;
vector < Pair <> > muted;
sendAll (from, msg) {...};
sendto(from, to, msg) {...};
register (Colleague c) {...};
(1...*)
User
sendAll (msg) {..};
sendto (to, msg) {..};
receive (from, msg) {..};
# standard UML Design
<<abstract>>
IMediator
add (colleague c);
send (string msg);
<<abstract>>
IColleague
IMediator mediator;
setMediator (m);
send (string m);
receive (string m);
Concrete Mediator
vector <Colleague> list;
add (colleague c) {..};
send (string msg) {..};
(1...*)
Concrete Colleague
setMediator (m) {..};
send (string m) {..};
receive (string m) {..};

# Why Mediator look same as Observer

- Observer Pattern : When one object (the subject) changes state, all its observers are notified and updated automatically.
  - Loosely coupled btw subject and observer

- Mediator Pattern : Centralizes communication btw colleagues (objects) through mediator
  ↳ Loosely coupling objects

# Standard Definition

- Defines an object that encapsulates how a set of objects interact, and promote loose coupling by preventing them from referring to each other.

# Real Life Use Case

① Chat Room
② Online Match