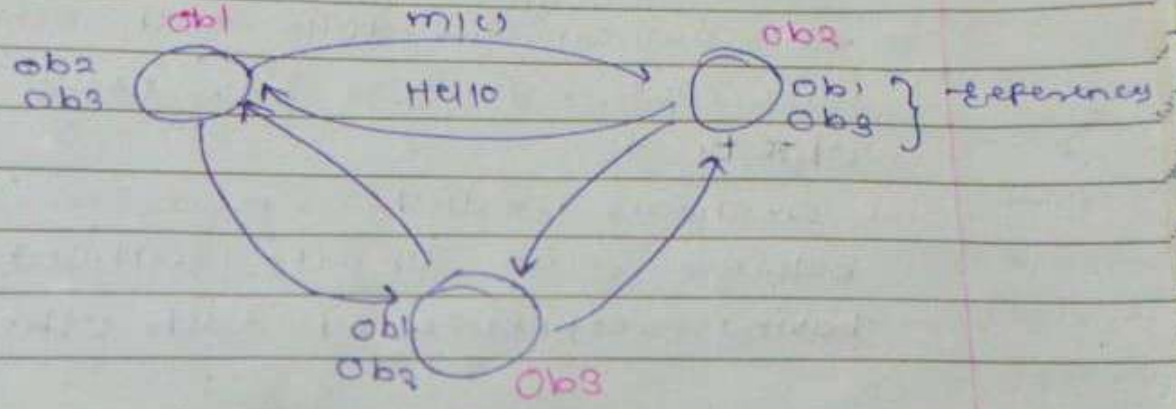
Lecture35:- Mediator Design pattern

Introduction

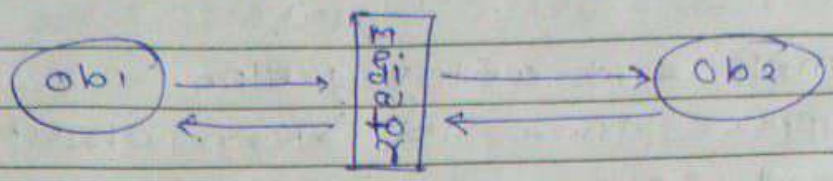
- Simple and widely used pattern
- Suppose there are two objects and they need to communicate, so to communicate or interact they both need reference of each other.



If we add ob3 then have to provide reference of ob3 to ob1 & ob2. In same way if we have n objects then we need (n-1) interaction. and same no. of reference we need to store whether it's list / vector. This makes it tightly coupled and breaks OCP principle too for every objects.

∴ Hence there comes entry of mediator pattern.

How mediator DP solves problem?



we create the mediator object which store the object references which want to communicate with each other. all objects will have references of only mediator object.

∴ In short, mediator performs communication between two objects without objects having reference of each other.

problem its solving → ① Loosely - coupled.
② Not too much reference in every object.

chatRoom example :- (without Mediator pattern)

① Multiple user interact with each other.
② Now we have list of users of all users in our class.
③ In chat Room, they can
ⓐ broadcast msg ie:- send to every member
ⓑ private member ie:- send to selective pattern.

User
vector<user> users; string name; sendAll (msg) {} send (msg, to){ } receive (msg, name) {}

④ SendAll (msg);
It store all users in vector which call receive() of all objects.

⑤ Send (msg, to);
It check receiver user name and send msg to it. and the receiver objects are store in mediator obj.

⑥ User class will not be singleton class as the users grow exponentially and object get created. Each New objects stores the object reference of all Another objects. It take too much memory.

⑦ And still it's manageable but what if we create complex methods to perform.

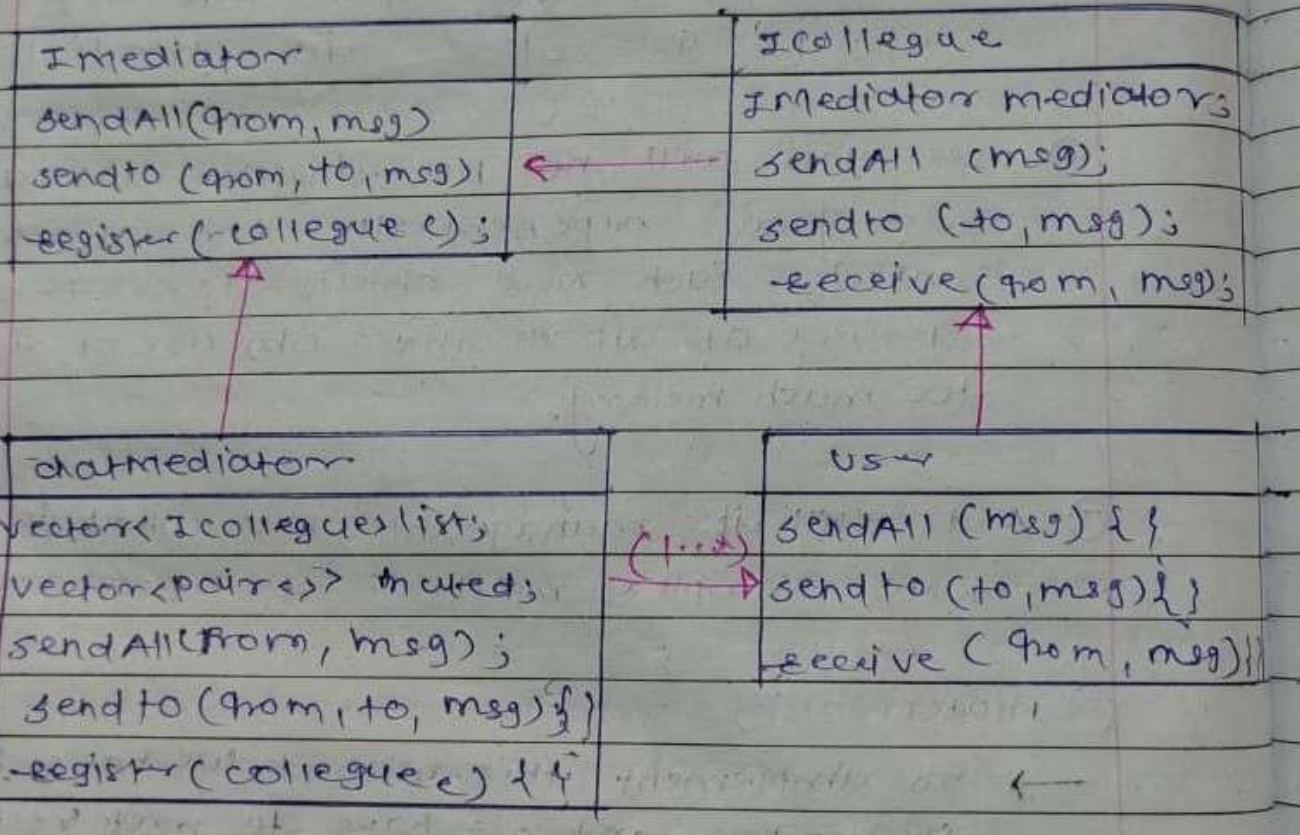
⑧ mute()
→ To implement this method we have store users in vector. and we have to create vector users, which are muted. if user is present in that list our task is to not notify about that msg.
So Now, to sendAll() we have to check.

```
for (auto user : users)                    ie:-
{ if(user → getmethod (name))    muted = {mohan}
  { continue; }                  mohan sends
  else { user → receive;         msg
  }
```

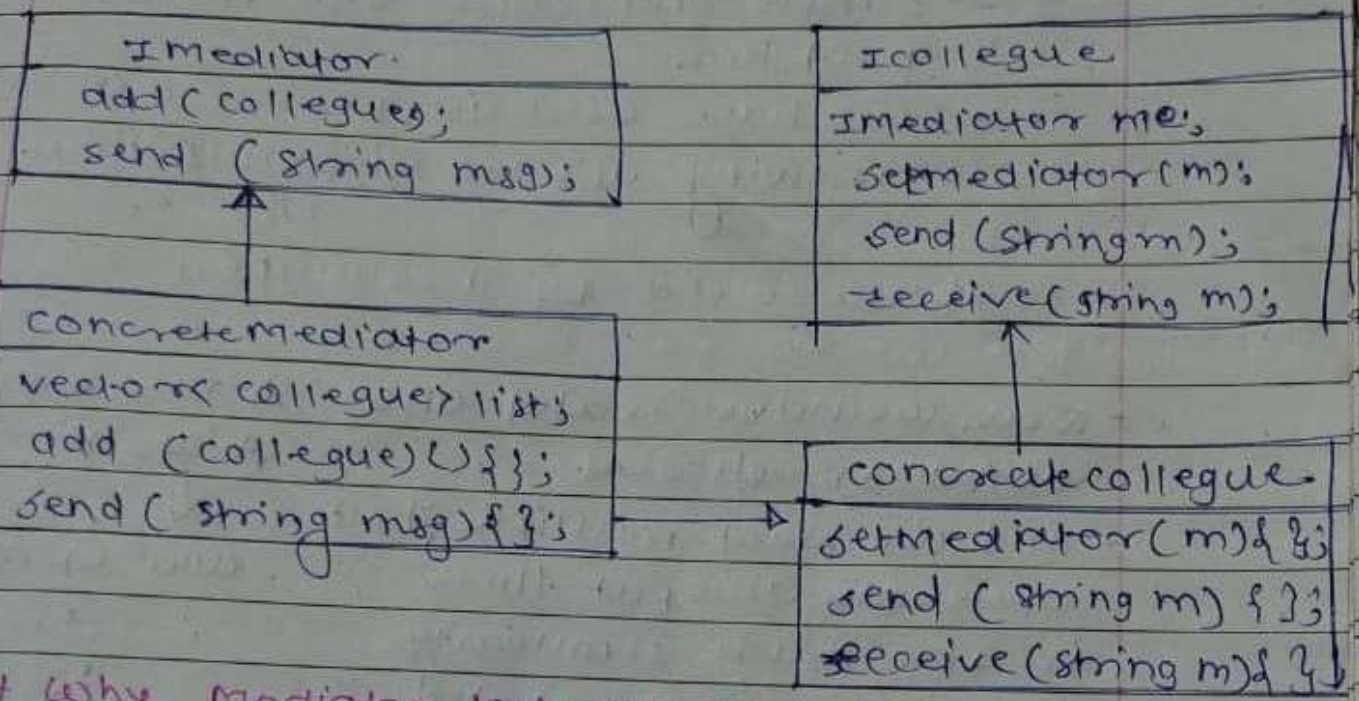
∴ Here solution is mediator pattern.

Solution of chat Room problem :- Mediator problem.

User will call collague as they don't have reference of each other but communicates This makes us loosely coupled.



standard uml diagram:-



Why Mediator look same as observer.

* Observer pattern :- when one object (the subject) changes state all it's observers are notified and updated. automatically.
- Loosely coupled b/w object and observer.

* Mediator pattern:-
centralizes communication between collegues (objects) through mediator
↳ Loosely coupling objects.

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