Mediator

Excuse me, did you say Observer?

NOPE!

PROBLEM

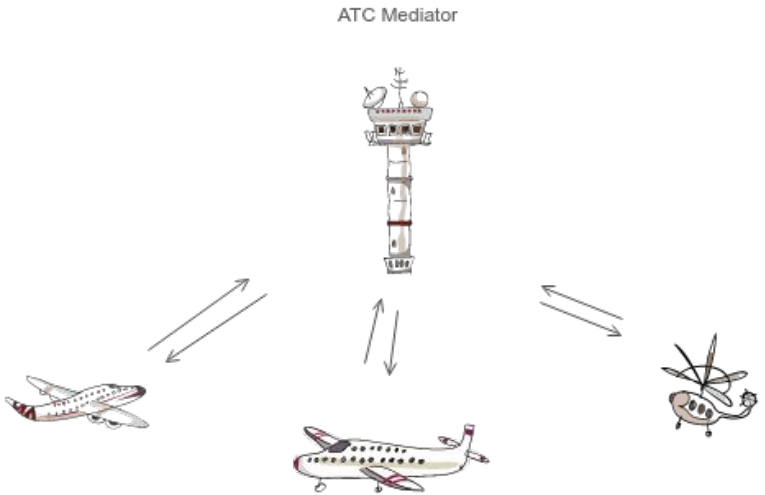
We want to design reusable components, but dependencies between the potentially reusable pieces demonstrates the "spaghetti code" phenomenon (trying to scoop a single serving results in an "all or nothing clump").

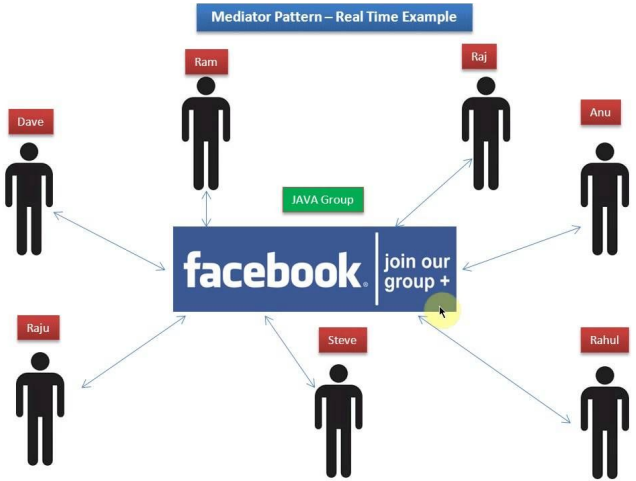

funny
DIE.COM

Mediator is here to save the day!

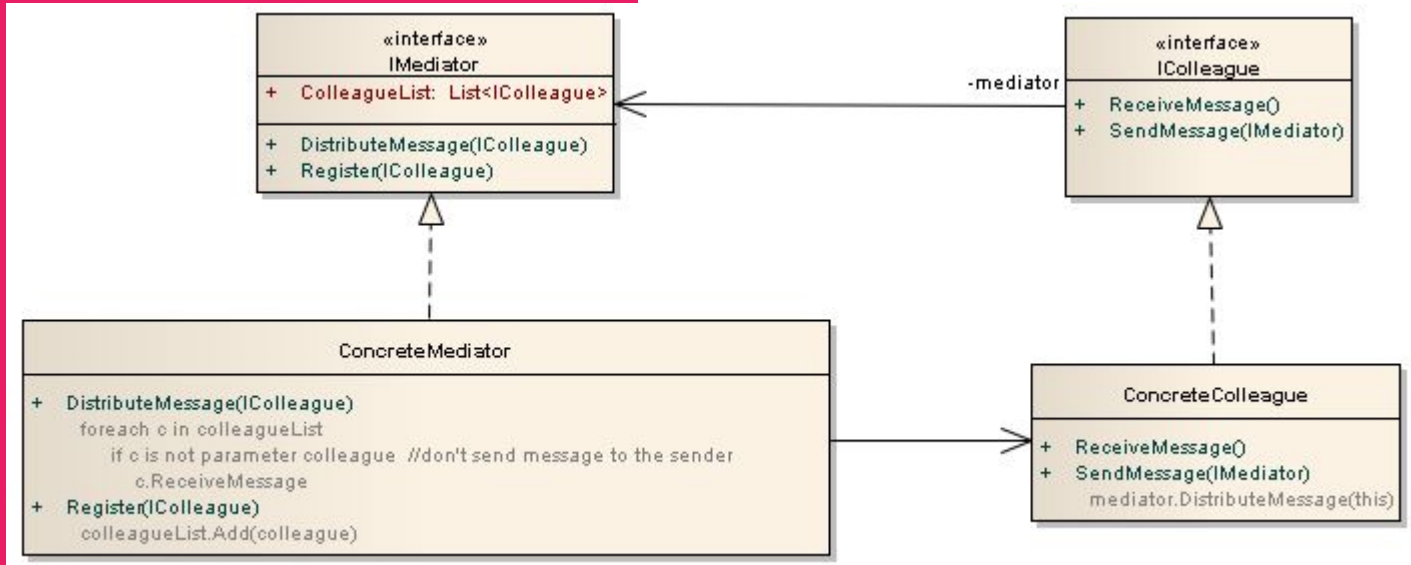
- Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
- Define an object that encapsulates how a set of objects interact.
- Design an intermediary to decouple many peers.
- Promote the many-to-many relationships between interacting peers to "full object status".

Bruh, do you even mediator?





UML DIAGRAM


«interface»
IMediator
+ ColleagueList: List<IColleague>
+ DistributeMessage(IColleague)
+ Register(IColleague)
-mediator
«interface»
IColleague
+ ReceiveMessage()
+ SendMessage(IMediator)
ConcreteMediator
+ DistributeMessage(IColleague)
foreach c in colleagueList
if c is not parameter colleague //don't send message to the sender
c.ReceiveMessage
+ Register(IColleague)
colleagueList.Add(colleague)
ConcreteColleague
+ ReceiveMessage()
+ SendMessage(IMediator)
mediator.DistributeMessage(this)

Pros - cons - any advice?

✓ Loosely-coupled(ness)

✓ Comprehension

✓ Simpler protocols

✓ Less subclassing

✗ Complexity

U sure u didn’t said Observer?

Well, just a little…

And maybe Facade too…

How was that again?

Kind of like this...

- CoR -> sender request along a chain of potential receivers.
- Command -> sender-receiver connection with a subclass.
- Observer -> very decoupled interface | multiple receivers to be configured at run-time.

- Mediator -> senders and receivers reference each other indirectly.

Let’s get ready to rumbleeeeeeee!!!

Round 1: Observer

- Observer -> Distributes communication by introducing "observer" and "subject" objects

  ---

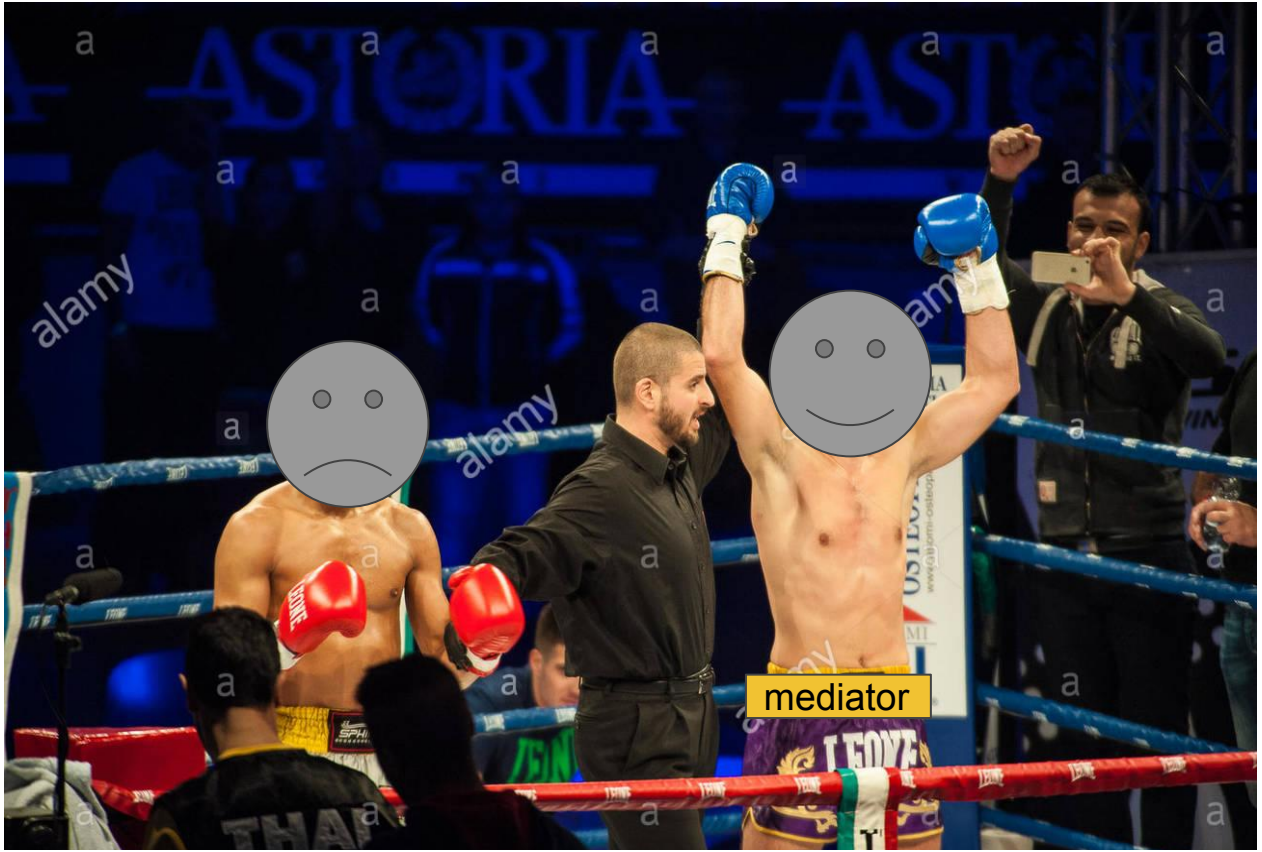
- Mediator -> Mediator object encapsulates the communication between other objects

Round 2: Facade

- Facade -> defines a simpler interface to a subsystem, it doesn't add new functionality, and it is not known by the subsystem classes

  ---

- Mediator -> abstracts/centralizes arbitrary communication between colleague objects, it routinely "adds value", and it is known/referenced by the colleague objects.


mediator

CODE

---

https://github.com/a-rmz/mediatorPattern

Further reading:
https://dzone.com/articles/design-patterns-mediator
https://sourcemaking.com/design_patterns/mediator
http://www.oodesign.com/mediator-pattern.html