24 – Mediator

# Setting

* Designing a simple chat room.

# Steps

## 01

* Walk through the existing solution. Show how registration does not all at scale.
* Run the example
  + Record the number of messages sent
* Add the Spammer
  + Record the number of messages sent. Does not even get heard, but…

## 01 🡪 02 Refactor to Mediator

* Create IMediator interface
  + void Register( IColleague )
  + void Distribute( IColleague sender, string messageContents )
* Create Mediator : IMediator class
  + private readonly ISet<IColleague>
  + **Ctor – initialize**
  + void Register( IColleague ) -> Add to \_colleagues set
  + void Distribute( IColleague sender, string messageContents ) ->
    - Send to everybody
    - **Except sender itself!**
* Modify IColleague
  + Remove: void Register() – Not needed! Handled by Mediator
* Modify Colleague
  + Add to **Ctor ( IMediator )**
  + Remove: void Register() – Not needed! Handled by Mediator
  + **Remove: IList<IColleague>** – Not needed! Handled by Mediator
  + Add: Send() should now call \_mediator.Distribute( this, messageContents )
* Modify Program.cs accordingly
* Run the example
  + Record the number of messages sent
* Add the Spammer
  + Record the number of messages sent. Does not get heard, but still no change…

## 02 🡪 03 Let mediator do the filtering!

* Modify IMediator interface
  + Add: void Register( IColleague**, params string[] topics** )
* Create ColleagueInfo
  + IColleague Colleague
  + IEnumerable<string> Topics
* Modify Mediator
  + Now use ColleagueInfo
  + Modify: void Register( IColleague**, params string[] topics** )
    - Trim before saving as ColleagueInfo
  + Modify: void Distribute()
    - Filter before sending out to clients

var interested = \_colleagues

    .Where(info => info.Colleague != sender)

    .Where(info => info.Topics.Intersect(words).Any()

.Select(info => info.Colleague));

* Modify Program.cs accordingly
* **Now run again:**
  + **3 messages only!!**