


# Behavioral Design Patterns

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

# Behavioral Patterns

- **Chain of Responsibility (CoR)**
- Command
- Interpreter
- Iterator
- **Mediator**
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor



# Chain of Responsibility (CoR)

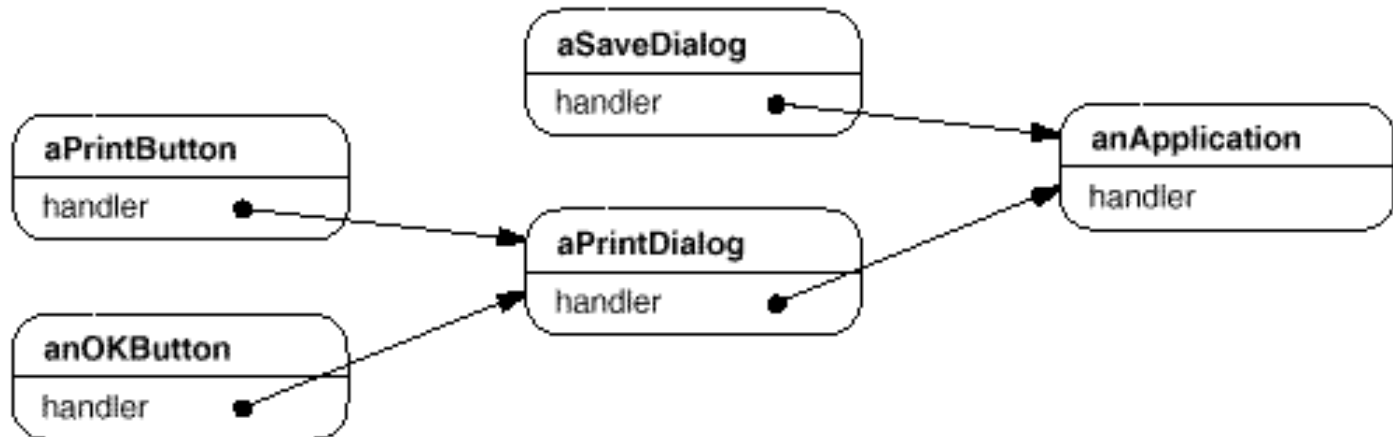
# [ Motivation ]

---

- Multiple objects can possibly handle a request
  - The object that handles is not known
- The request gets passed along a chain of objects until one of them handles it
  - An object in the chain either handles the request or forwards it to the next object

# Example of CoR

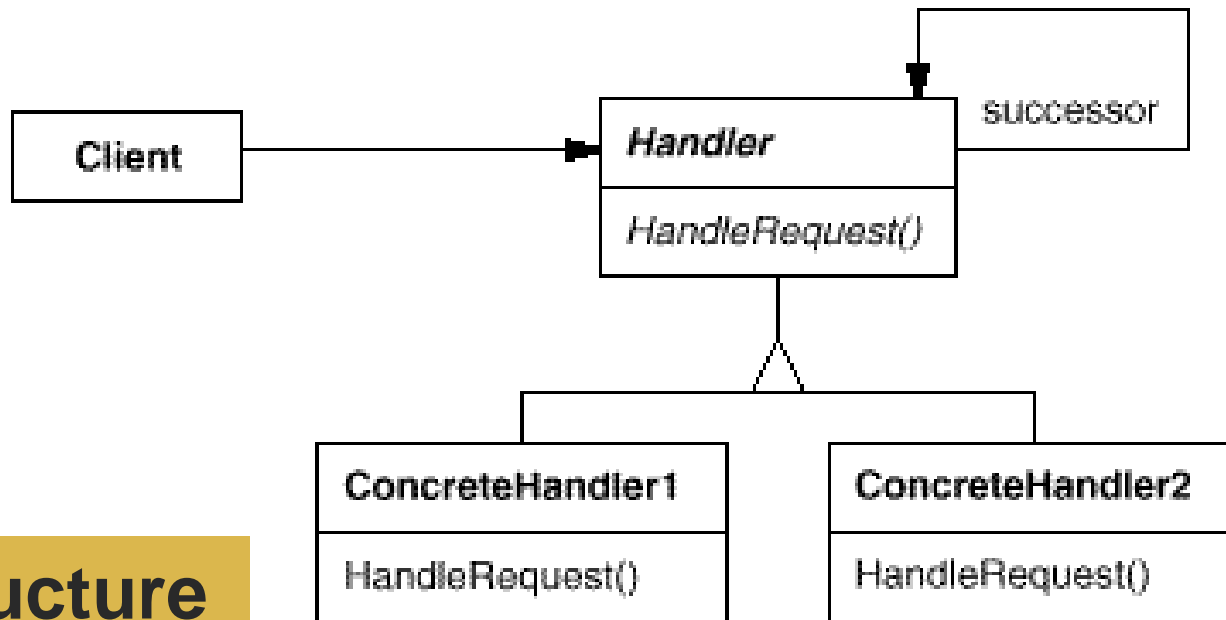
- Let's consider a context-sensitive help facility for a graphical user interface (GUI)
  - A button may explain how it works
  - If not, a dialog can provide a higher-level help
  - If not, the application gives a general help



# [ Chain of Responsibility ]

- Name
  - Chain of Responsibility (CoR)
- Problem Description
  - To give more than one object the chance to handle a request
- Solution (*next slide*)
- Consequences
  - Avoid coupling the sender to its receiver
  - Flexibility to add new handler objects

# CoR Solution

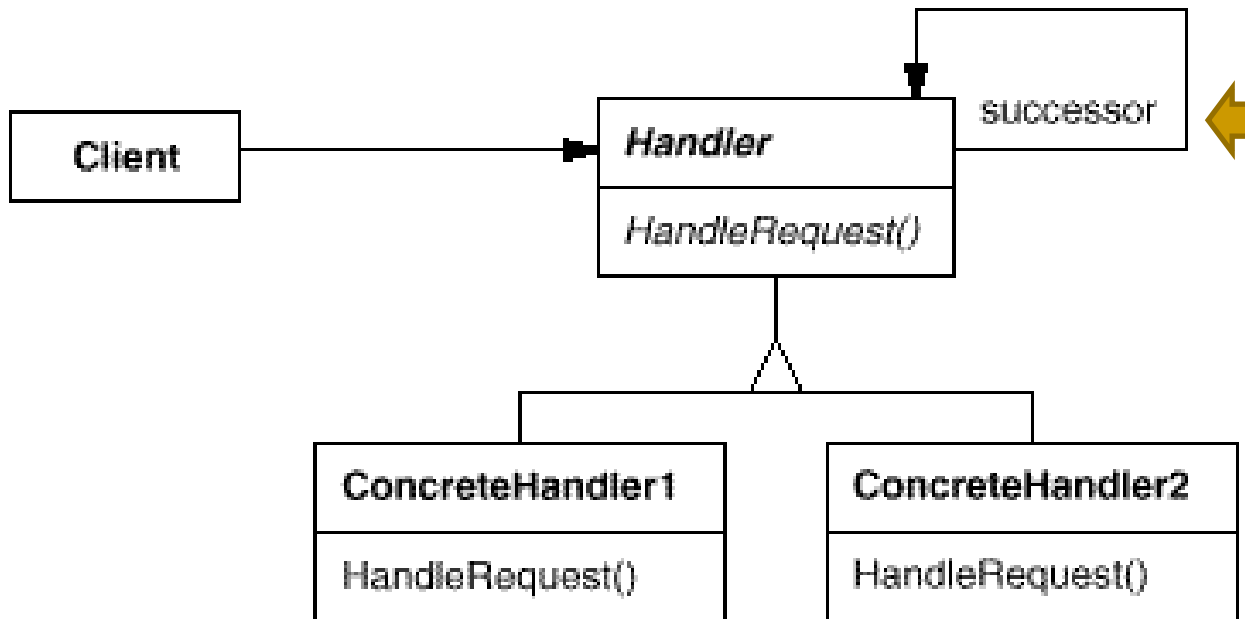


## Structure



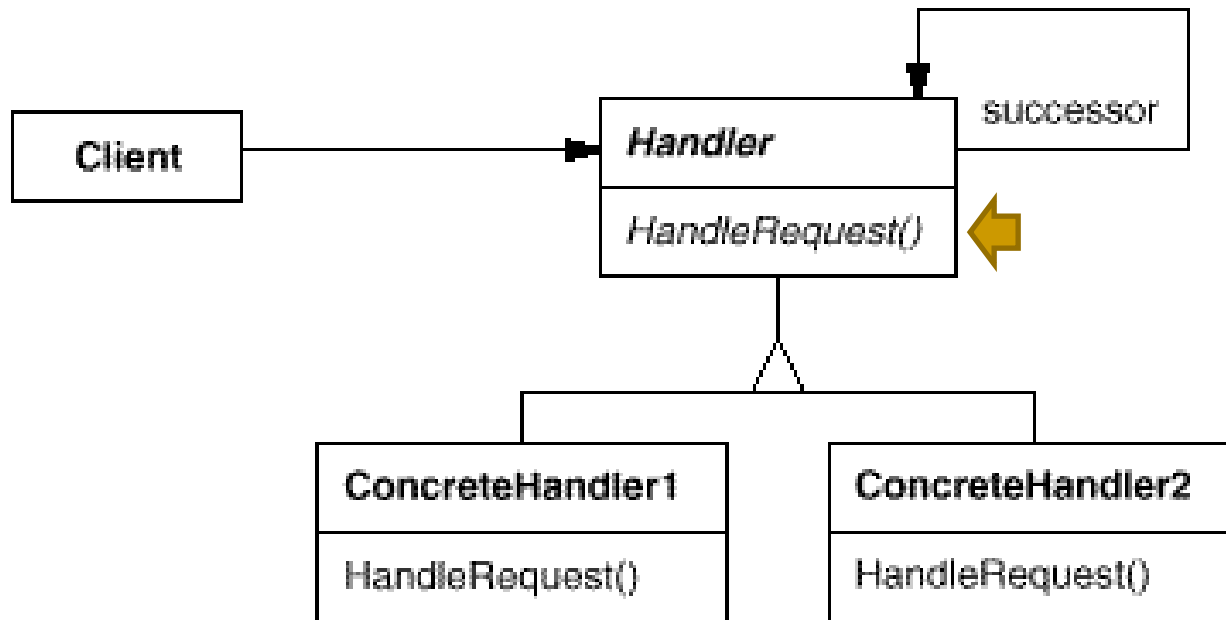
## Behavior

# Successor in the Chain



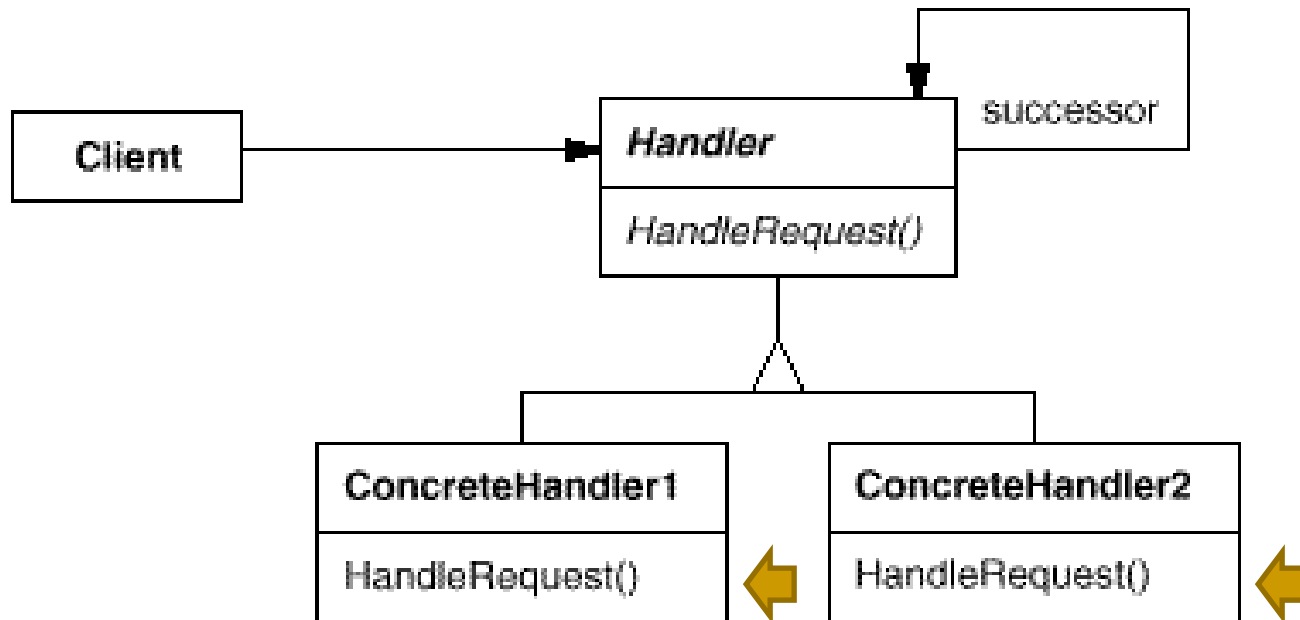
**Every handler has a reference to the next object in the chain.**

# [ The Handler Class ]



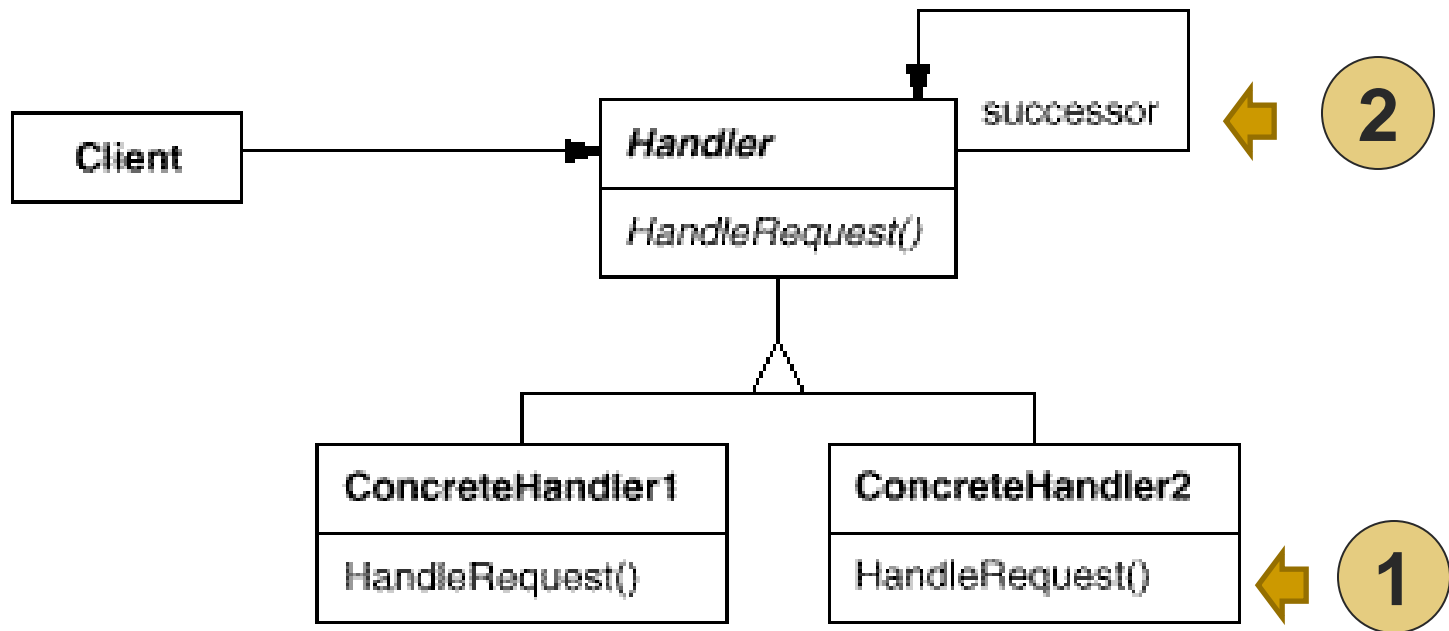
**The abstract Handler class has a method signature to handle the request.**

# [ The ConcreteHandler Classes ]



**Each ConcreteHandler class implements a specific handle method.**

# [ Object Options ]



**A handler object has two options:**

- 1. Handle the request or**
- 2. Forward the request to the next handler.**



Mediator

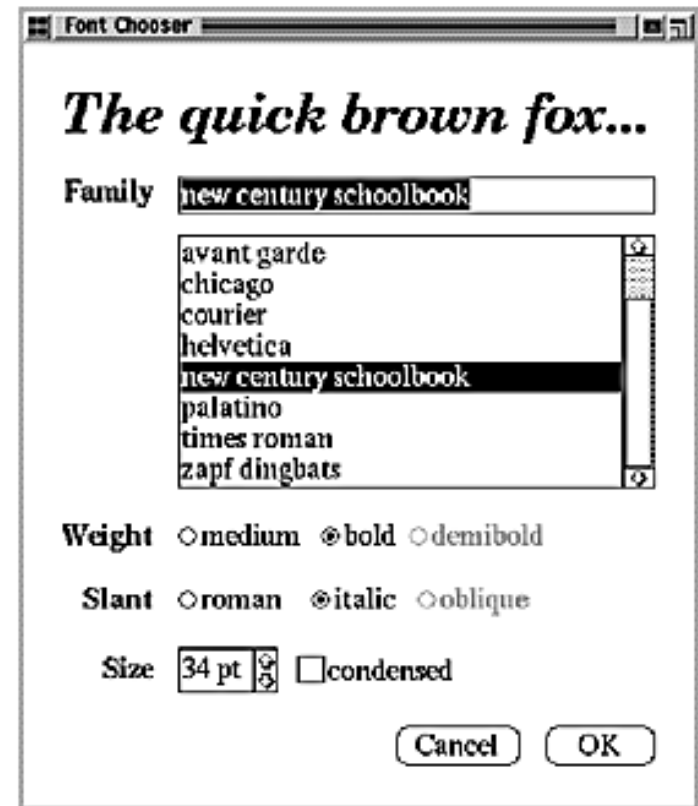
# [ Motivation ]

---

- Object-oriented design encourages the distribution of behavior among objects
  - Objects have to interact with other objects
  - In the worst case, every object ends up knowing all other objects
- Lots of coupling connections make it difficult to change the system

# Example

- Let's consider widgets in a GUI
- For instance
  - A button is disabled when a field is empty
  - Selecting an entry in a list of choices might change the content of a text field

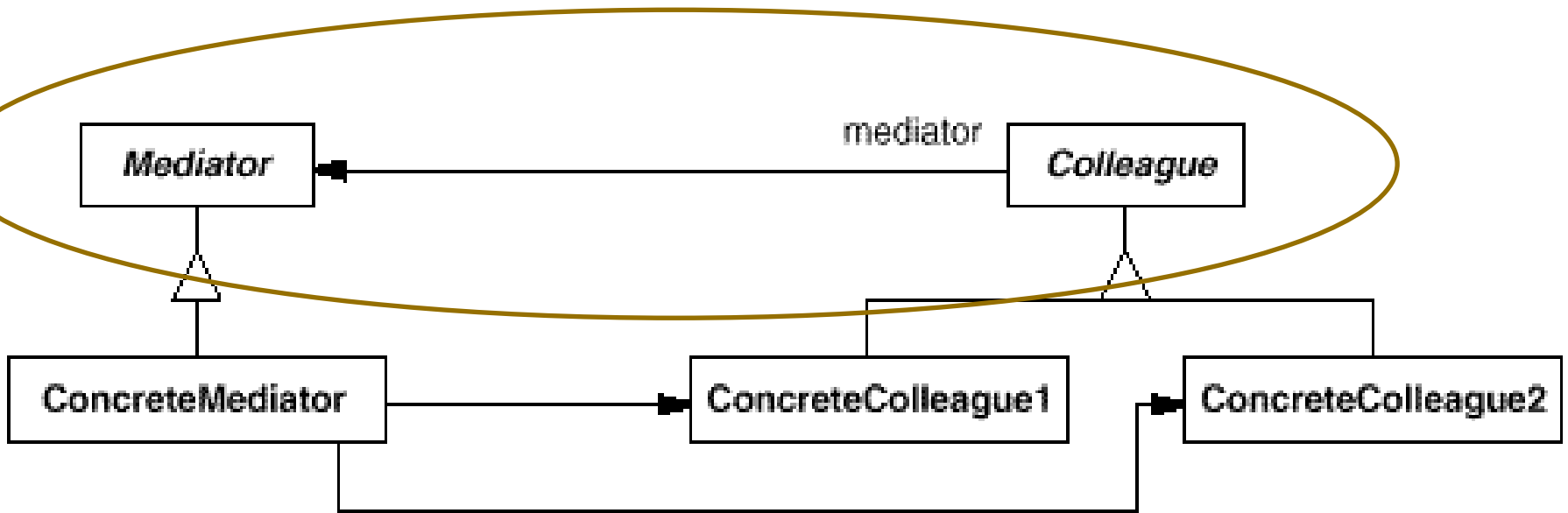


# [ Mediator ]

---

- Name
  - Mediator
- Problem Description
  - To define an object that encapsulates how a set of objects interact
- Solution (*next slide*)
- Consequences
  - It decouples colleague objects
  - It centralizes control

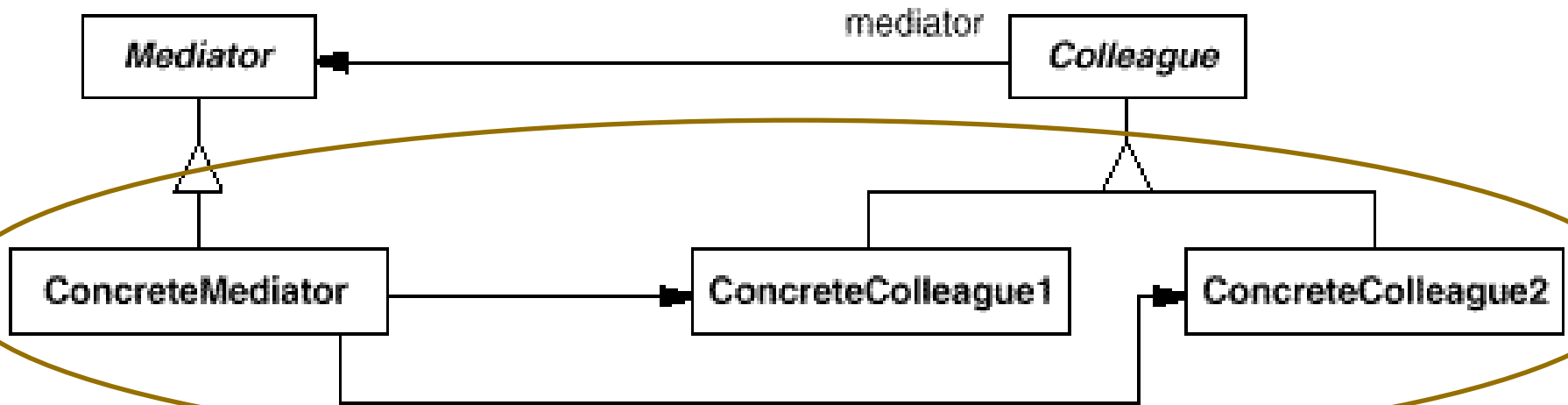
# [ Mediator Solution ]



**A Colleague class knows the Mediator class.**

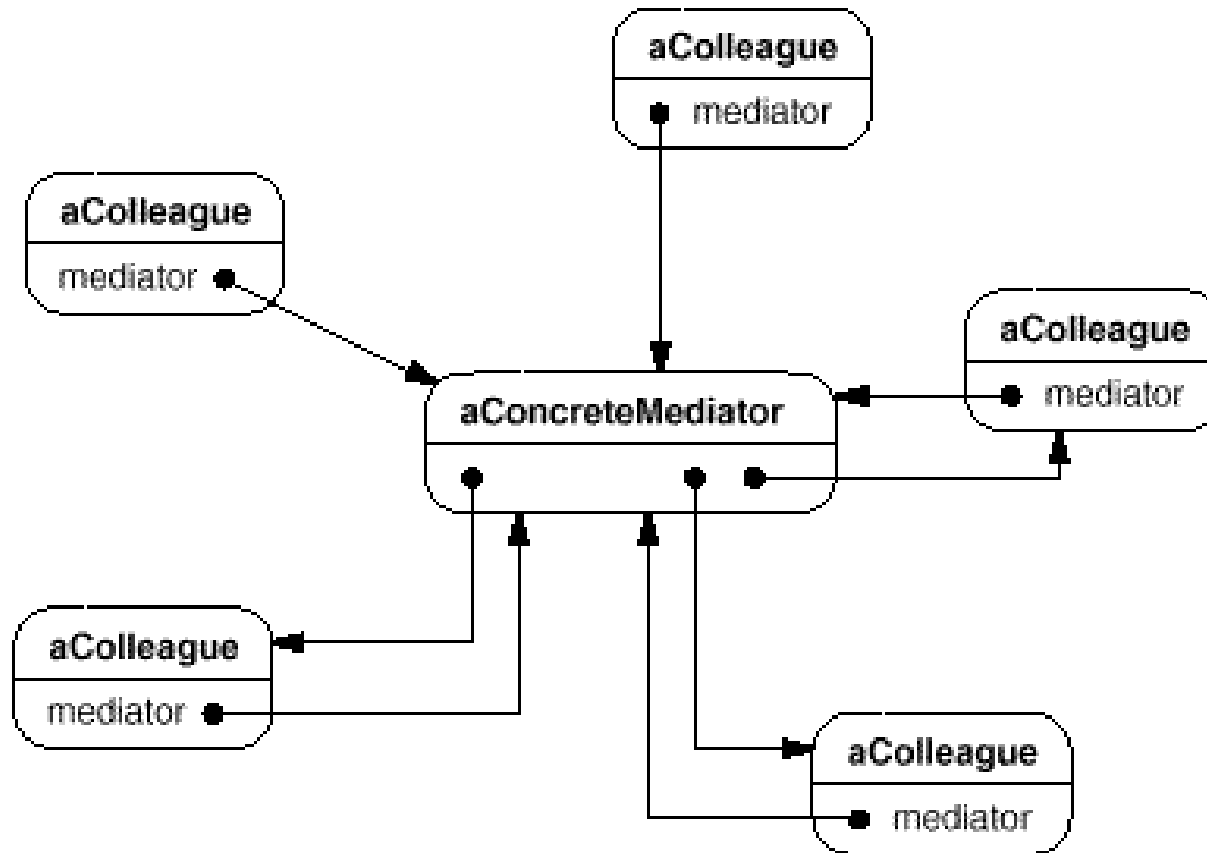
**ConcreteColleague classes interact with the Mediator by means of this relationship.**

# [ ConcreteMediator ]



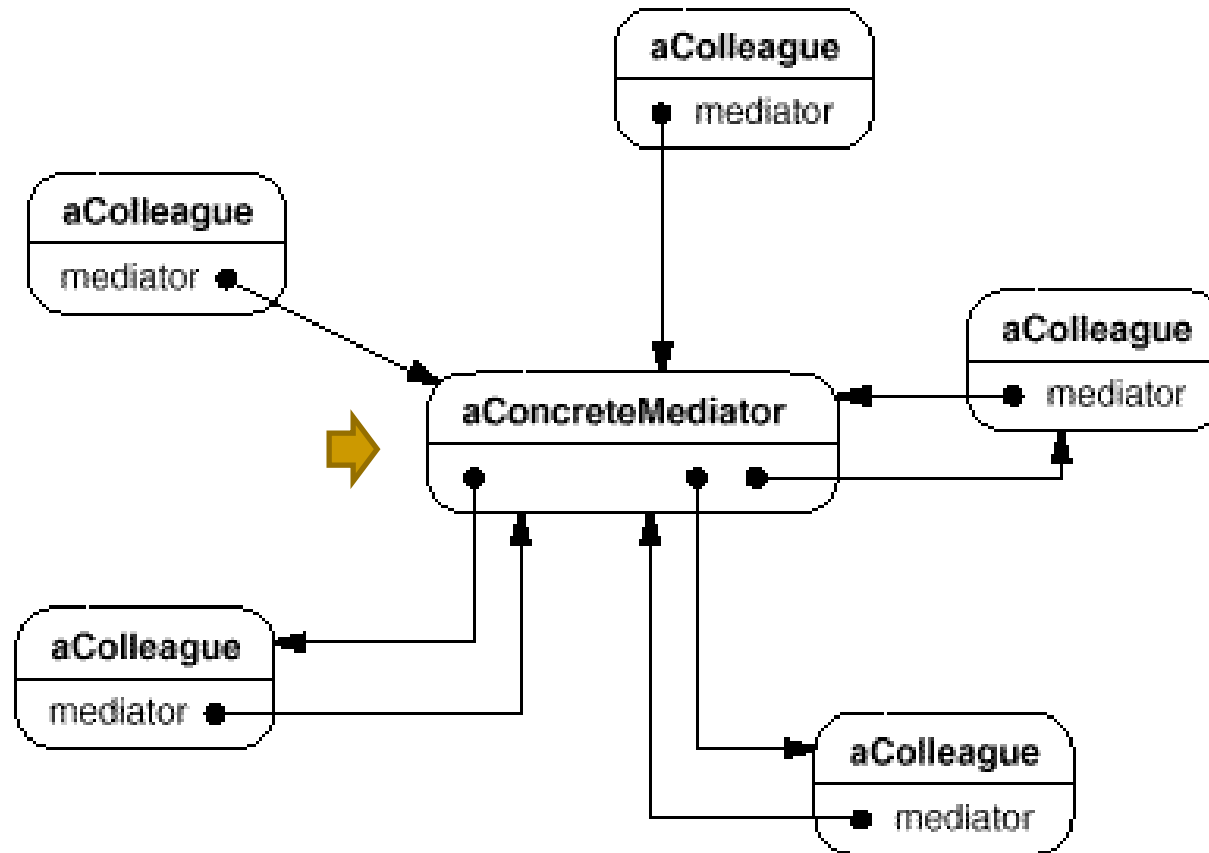
**The ConcreteMediator knows all ConcreteColleague classes.**

# Behavioral Collaboration



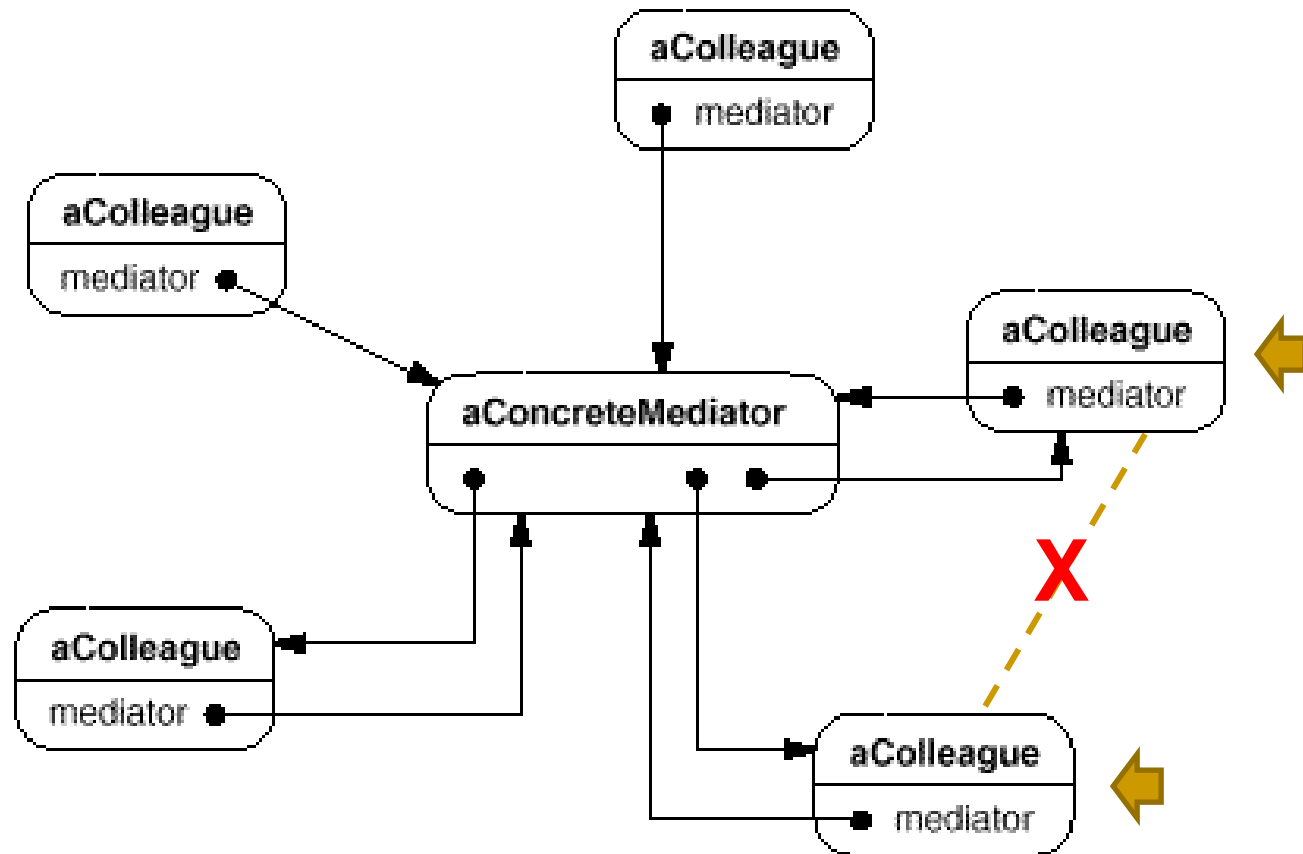
**The Mediator knows Colleague objects.  
All Colleague objects know the Mediator.**

# The Mediator Role



**A mediator replaces many-to-many interactions by one-to-many interactions.**

# Colleague Communication



**A Colleague object cannot call another colleague directly**

# [ Bibliography ]

- E. Gamma, R. Helm, R. Johnson, J. Vlissides. **Design Patterns**, 1st. Edition. Addison Wesley, 1994.
  - Chain of Responsibility and Mediator