Mediator, Decorator & Chain of Responsibility

Alessandro Midolo, Ph.D. Student
alessandro.midolo@phd.unict.it

Tutorato Ingegneria del Software

A.A. 2021/2022

Mediator

Intento

- Definire un oggetto che incapsula come un gruppo di oggetti interagisce. Promuove il lasco accoppiamento fra oggetti poiché evita che essi interagiscano direttamente, e permettere di modificare le loro interazioni indipendentemente da essi

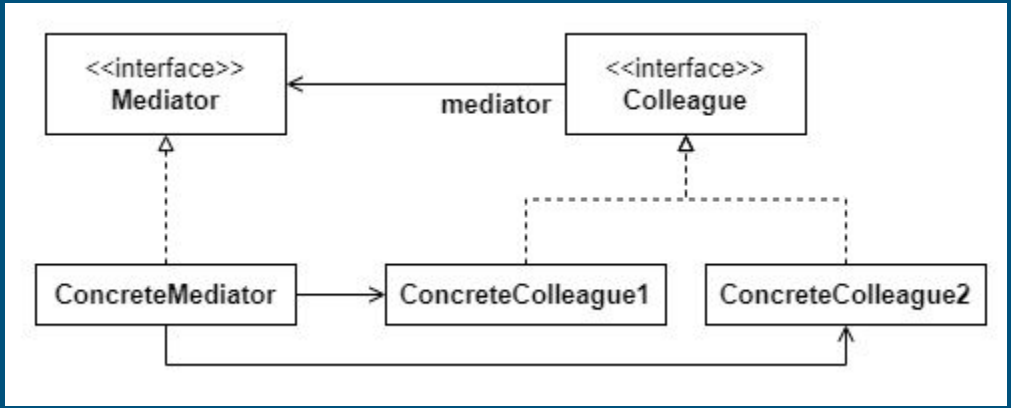
Problema

- La distribuzione di responsabilità fra vari oggetti può risultare in molte connessioni tra oggetti → un oggetto conosce tutti gli altri
- Molte connessioni rendono un oggetto dipendente da altri. Inoltre potrebbe essere difficile cambiare il comportamento del sistema poiché distribuito tra vari oggetti

Soluzione

- **Mediator** definisce un'interfaccia per gli oggetti connessi, **Colleague**
- **ConcreteMediator** implementa il comportamento cooperativo e coordina i **Colleagues**
- **Colleague** conosce il **Mediator** e comunica con esso quando avrebbe comunicato con un altro **Colleague**
- **ConcreteColleague** mandano e ricevono richieste a un oggetto **Mediator**, il quale inoltra le richieste agli altri **CC**

Diagramma UML Mediator



Conseguenze Mediator

- La maggior parte della complessità per la gestione delle dipendenze è spostata dagli oggetti cooperanti al *Mediator* → gli oggetti sono più facili da implementare e mantenere
- Le classi *Colleague* sono più riusabili poiché la loro funzionalità fondamentale non è mischiata con il codice che gestisce le dipendenze
- Il codice del *Mediator* non è in genere riusabile poiché la gestione delle dipendenze implementata è specifica della applicazione

Decorator

Intento

- Aggiungere ulteriori responsabilità ad un oggetto dinamicamente. I decorator forniscono un'alternativa flessibili all'implementazione di sottoclassi per estendere funzionalità

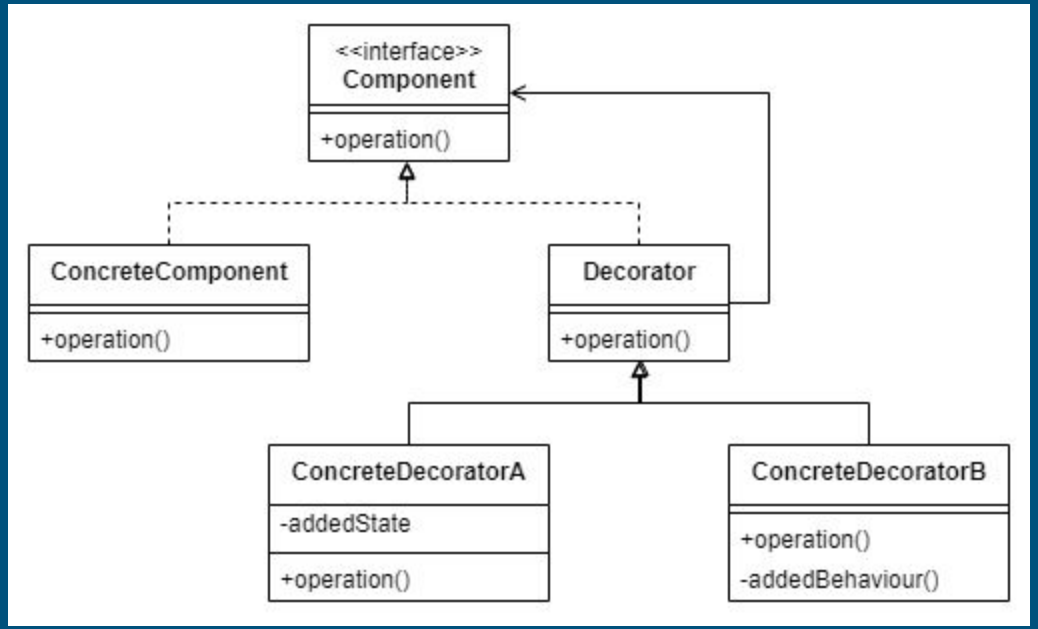
Problema

- Alcune volte si vogliono aggiungere responsabilità a singoli oggetti e non all'intera classe. Si potrebbe usare l'ereditarietà, ma così un client non può controllare come e quando "decorare" un componente
- A volte la creazione di classi non è praticabile. Un numero grande di estensioni produrrebbe un numero enorme di sottoclassi per gestire tutte le combinazioni

Soluzione

- **Component** definisce l'interfaccia per gli oggetti che possono avere aggiunte le responsabilità dinamicamente
- **ConcreteComponent** definisce un oggetto su cui poter aggiungere responsabilità
- **Decorator** mantiene un riferimento a un oggetto *Component* e definisce un'interfaccia conforme a quella di *Component*. Inoltra le richieste al suo oggetto *Component* e può fare altre operazioni prima e dopo l'inoltro
- **ConcreteDecorator** implementa la responsabilità aggiunta al *Component*

Diagramma UML Decorator



Conseguenze Decorator

- Più flessibilità rispetto all'ereditarietà poiché si possono aggiungere responsabilità dinamicamente
- La stessa responsabilità può essere aggiunta più volte → aggiungendo due istanze dello stesso *ConcreteDecorator*
- Prevedere per le classi in alto nella gerarchia tutte le responsabilità che servono significherebbe avere per esse troppe responsabilità → i *ConcreteDecorator* sono indipendenti e permettono di aggiungere responsabilità successivamente
- Si avranno tanti piccoli oggetti che differiscono nel modo in cui sono interconnessi

Chain of Responsibility

Intento

- Evitare di accoppiare il mandante di una richiesta con il suo ricevente dando la possibilità di gestire la richiesta a più oggetti. Concatena gli oggetti riceventi e passa la richiesta attraverso la catena fin quando un oggetto non la gestisce

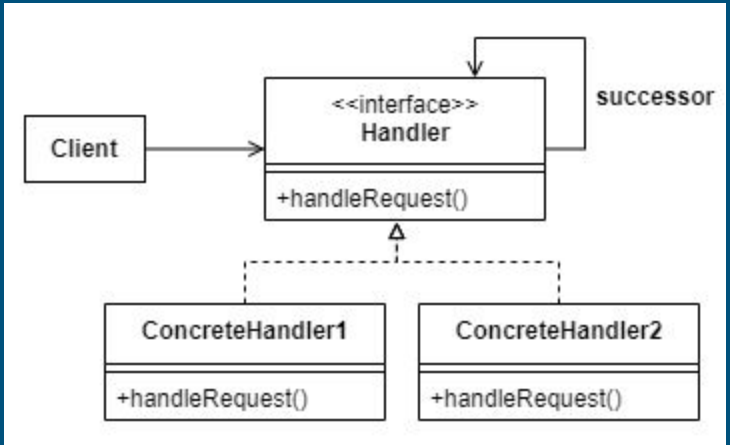
Problema

- Si vuole effettuare una richiesta a diversi oggetti senza specificare il ricevente in modo esplicito
- Più di un oggetto può gestire la richiesta e il ricevente non è conosciuto a priori, questo deve essere stabilito automaticamente
- L'insieme di oggetti che possono gestire le richieste dovrebbero essere specificati dinamicamente

Soluzione

- **Handler** definisce un'interfaccia per gestire le richieste e può avere un riferimento al successore
- **ConcreteHandler** gestisce le richieste per cui è responsabile. Ha un riferimento al suo successore → se non può gestire la richiesta la inoltra al successore

Diagramma UML COR



Conseguenze COR

- L’oggetto che effettua la richiesta non sa chi la gestirà, sa solo che verrà gestita in modo appropriato. Sia il ricevente che il mittente non si conoscono, e un oggetto della catena non conosce la struttura di essa → minore dipendenza tra gli oggetti
- E’ possibile aggiungere o cambiare le responsabilità per gestire una data richiesta cambiando la catena a run-time
- Una richiesta potrebbe essere attraversare tutta la catena senza essere propriamente gestita → se la catena non è configurata correttamente, la richiesta potrebbe rimanere non gestita