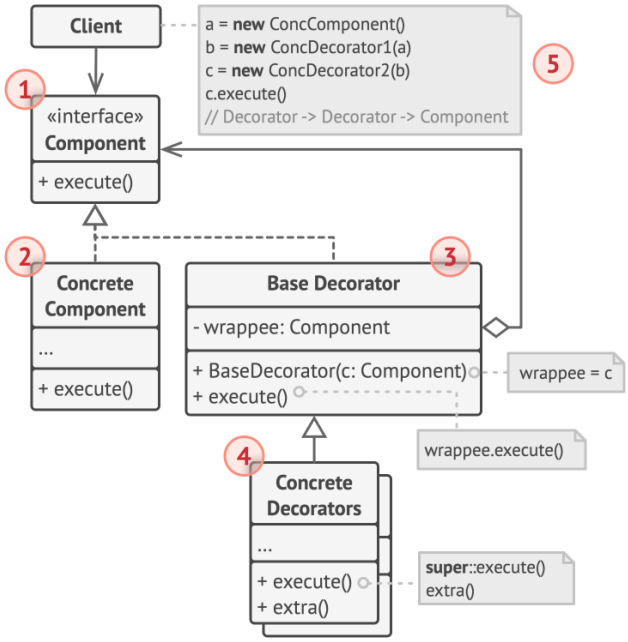
Decorator

E’ un pattern strutturale che consente di associare nuovi comportamenti agli oggetti inserendo tali all’interno di speciali oggetti wrapper che contengono i comportamenti.

Wrapper è il soprannome alternativo del pattern, che esprime l’idea principale del pattern. Un Wrapper è un oggetto che puo essere collegato a un oggetto target. Il wrapper contiene lo stesso set di metodi del target e delega a quest’ultimo tutte le richieste che riceve. Ma esso può alterare il risultato eseguendo un’operazione prima o dopo aver passato la richiesta al target.

Il wrapper implementa la stessa interfaccia dell’oggetto wrappato.

Struttura



- Il componente dichiara l’interfaccia comune sia per i wrapper che per gli oggetti wrapper;
- Concrete component è una classe di oggetti che vengono avvolti. Definisce il comportamento di base, che può essere modificato dai decoratori;

- La classe Base Decorator ha un campo per fare riferimento a un oggetto wrapper. Il tipo del campo deve essere dichiarato come interfaccia del componente in modo che possa contenere sia componenti concreti che decoratori.
- I decoratori concreti definiscono comportamenti aggiuntivi che possono essere aggiunti dinamicamente ai comportamenti.
- Il client può racchiudere i comportamenti in più livelli di decoratori, purchè funzioni con tutti gli oggetti tramite l'interfaccia del componente.

Applicabilità

- Usare il pattern quando è necessario poter assegnare comportamenti aggiuntivi agli oggetti in fase di esecuzione senza interrompere il codice che usa tali oggetti;
- Usare il modello quando risulta complicato o non è possibile estendere il comportamento di un oggetto tramite ereditarietà.

Vantaggi

- E' possibile estendere il comportamento di un oggetto senza creare una nuova sottoclasse;
- E' possibile aggiungere o rimuovere responsabilità da un oggetto in fase di esecuzione;
- E' possibile combinare diversi comportamenti racchiudendo un oggetto in più decoratori;
- Principio di singola responsabilità.

Svantaggi

- E' difficile rimuovere il wrapper specifico dalla pila dei wrapper;

- E’ difficile implementare un decoratore in modo che il suo comportamento non dipenda dall’ordine nello stack dei decoratori;
- Il codice di configurazione iniziale dei livelli potrebbe apparire piuttosto brutto.

Relazione con altri pattern

- Adapter fornisce un’interfaccia diversa per accedere ad un oggetto esistente. Decorator ha un’interfaccia invariata o estesa e supporta la composizione ricorsiva;
- Con Adapter si accede a un oggetto esistente tramite un’interfaccia diversa. Con Proxy, l’interfaccia rimane la stessa. Con Decorator si accede all’oggetto tramite un’interfaccia migliorata;
- Chain of responsability e Decorator hanno strutture simili. Entrambi si basano sulla composizione ricorsiva per far passare l’esecuzione attraverso una serie di oggetti. Ma presentano differenze cruciali;
- Composite e Decorator hanno strutture simili poichè entrambi si basano sulla composizione ricorsiva per organizzare un numero aperto di oggetti. Decorator ha un solo componente figlio e aggiunge responsabilità aggiuntive all’oggetto incapsulato, mentre composite si limita a sommare i risultati dei suoi figli;
- I progetti che fanno ampio uso di Composite e Decorator possono spesso trarre vantaggio dall’uso di Prototype.
- Decorator ti consente di cambiare l’aspetto di un oggetto, mentre Strategy ti consente di cambiarne l’interno;
- Decorator e Proxy hanno struuttre molto simili, ma intenti diversi. Entrambi si basano sul principio di composizione ricorsiva, ma Proxy gestisce autonomamente il ciclo di vita del suo oggetto di servizio, mentre Decorator è sempre controllata dal client.