## Decorator

### Kurzbeschreibung

Beim Decorator-Muster wird eine konkrete Komponente um beliebig viele konkrete Dekorierer erweitert. Die konkrete Komponente und der Dekorierer implementieren beide den Supertyp Komponente. Konkrete Dekorierer implementieren den Supertyp Dekorierer. Konkrete Dekorierer enthalten eine Instanzvariable für die Komponente, die sie dekorieren. Beim Dekorieren erweitern Sie den Zustand der gespeicherten Variable, indem sie vor oder nach Abruf dieses Zustandes ihr eigenes Verhalten hinzufügen. Den konkreten Dekorierern ist es egal, ob sie eine konkrete Komponente oder eine bereits dekorierte Komponente erweitern. [HD10, FFSB06]

**GoF-Definition**

"Erweitere ein Objekt dynamisch um Zuständigkeiten. Dekorierer bieten eine flexible Alternative zur Unterklassenbildung, um die Funktionalität einer Klasse zu erweitern." ([GoF96], Seite 199)

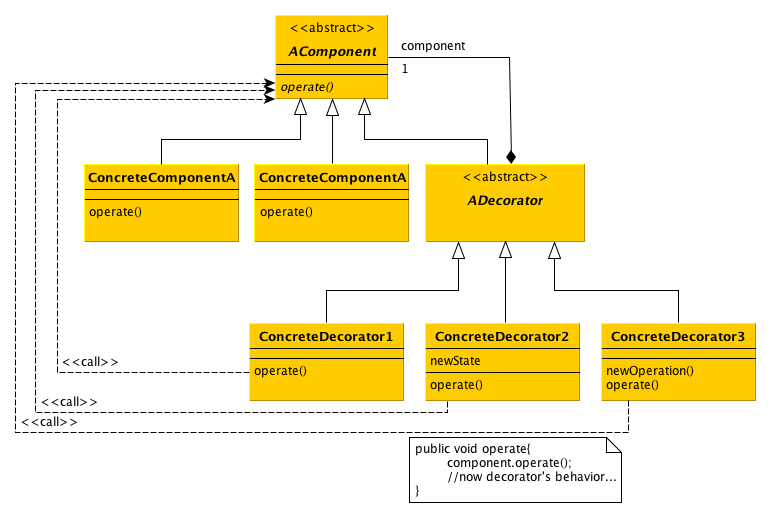


Abbildung 1: UML-Diagramm zum Decorator-Muster *(aus:* [HD10])

### Design-Prinzipien

* Veränderlichen Code kapseln
* Programmieren auf Schnittstellen
* Komposition der Vererbung vorziehen
* Offen/Geschlossen-Prinzip

### Vorteile

* Klassen können ohne Vererbung um Verhalten erweitert werden
* Nicht-dekorierter Komponenten und beliebig oft dekorierte Komponenten sind gleichermaßen vom Client verwendbar und nicht unterscheidbar
* Ein neuer konkreter Dekorierer kann einfach hinzugefügt werden, indem dieser den Supertyp Dekorierer implementiert

[HD10]

### Nachteile

* Durch Dekorieren mit vielen konkreten Dekorierer wird die Objektanzahl erhöht
* Die Komplexität steigt durch Verschachtelung vieler Komponenten an
* Die Fehlersuche wird durch Verwendung vieler konkreter Dekorierer erschwert

[HD10]

## Observer

### Kurzbeschreibung

Das Observer-Muster wird eingesetzt, wenn mehrere Observer von einem Observable (Eins-zu-viele-Beziehung) benachrichtigt werden sollen. Das kann notwendig sein, wenn sich z.B. sein Zustand geändert hat. Die Observer können sich beim Observer an- und wieder abmelden durch entsprechende Methoden (von einer Schnittstelle vorgeschrieben). Das Observable benachrichtigt die Observer über eine update-Methode (von einer Schnittstelle vorgeschrieben). Die Informationen können vom Observable zum Observer entweder nach dem Pull-Ansatz (Observer fragt nach und gibt Daten an, die er haben möchte) oder nach dem Push-Ansatz (Alle Daten werden zu den Observern übertragen) übermittelt werden. [HO10, FFSB06]

**GoF-Definition**

"Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, das alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden." ([GoF], Seite 287)

### UML-Klassendiagramm

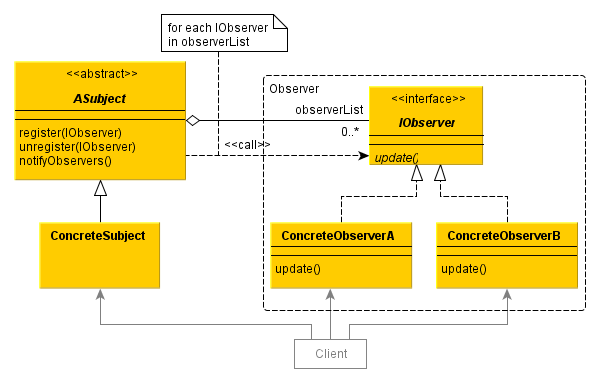


Abbildung 2: UML-Diagramm zum Observer-Muster *(aus:* [HO10])

### Design-Prinzipien

* Veränderlichen Code kapseln
* Programmieren auf Schnittstellen
* Komposition der Vererbung vorziehen

[FFSB06]

### Vorteile

* Die Zustände bleiben konsistent, da Observer ihr Verhalten bei Änderungen anpassen
* Ein Observer kann sich jederzeit (zur Laufzeit) beim Observable an- und wieder abmelden
* Ein Observable kann mehrere Observer benachrichtigen, ein Observer kann sich bei mehreren Observables registrieren
* Observable und Observer sind lose gekoppelt, da sie sich nur über Interfaces und somit nicht genauer kennen müssen

[HO10]

### Nachteile

* Jeder Observer muss sich beim Oberservable an- und wieder abmelden
* Bei umfangreichen Systemen kann eine kleine Änderungen zu großen Aktualisierungskaskaden führen

[HO10]

## Abstract Factory

### Kurzbeschreibung

Beim Abstract-Factory-Muster werden eine oder mehrere Familien zusammenhängender Produkte definiert. Die abstrakte Fabrik bietet die Schnittstelle, welche von allen konkreten Fabriken implementiert werden muss. Jede konkrete Fabrik kann ein konkretes Produkt einer Produktfamilie produzieren. Der Client nutzt als Schnittstelle nur die abstrakte Fabrik, um ein Objekt zu erlangen. [FFSB06, HAF10]

**GoF-Definition**

"Biete eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen." ([GoF96], Seite 107)

### UML-Diagramm

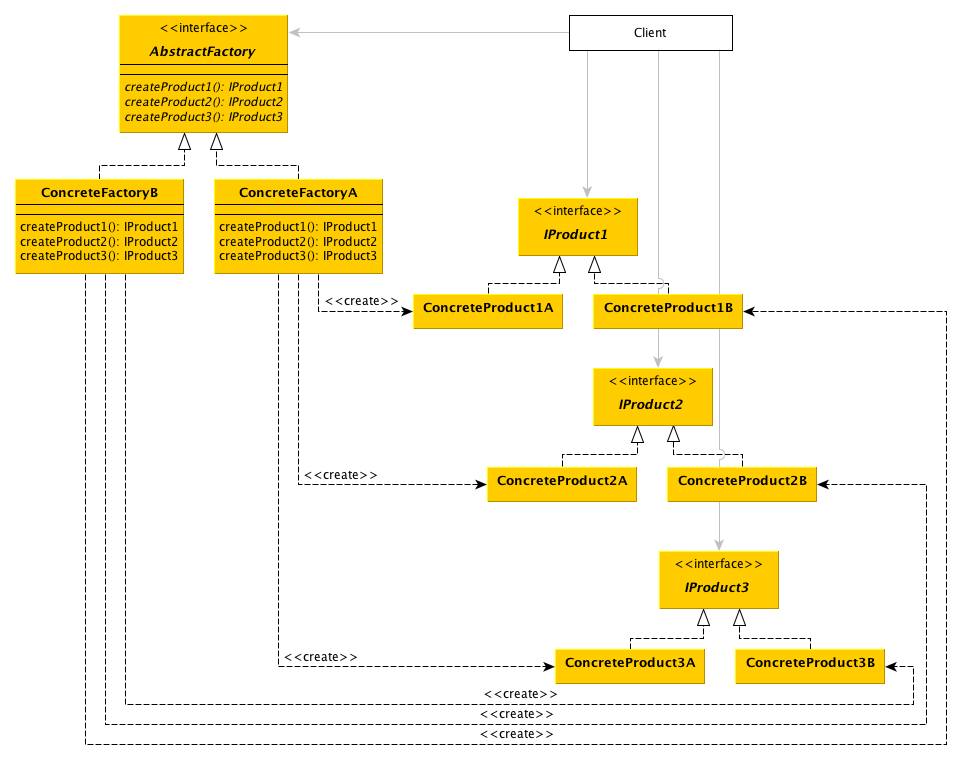


Abbildung 3: UML-Diagramm zum Abstract-Factory-Muster *(aus:* [HFM10])

### Design-Prinzipien

* Veränderlichen Code kapseln
* Programmieren auf Schnittstellen
* Offen/Geschlossen-Prinzip
* Umkehrung der Abhängigkeiten

### Vorteile

* Initialisierung eines Objekts von konkreter Implementierung getrennt
* Es können jederzeit neue Objekt- bzw. Produktfamilien hinzugefügt/entfernt werden
* Konkrete Produkte können in mehreren Produktfamilien vorkommen

### Nachteile

* Beim Hinzufügen neuer Produkte müssen die Abstract-Factory und alle konkreten Factorys modifiziert oder durch neue ersetzt werden.