Jindra Michael

Design Patterns

SEW Protokoll

Michael Jindra
28.11.2017

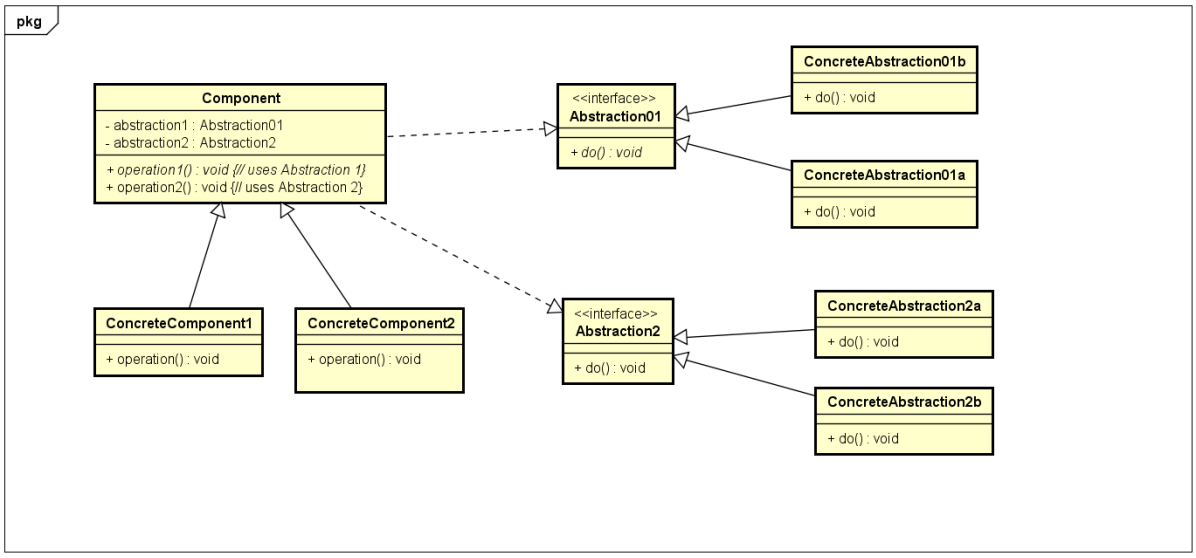
Design Patterns

Strategy

Warum Strategy?

Das Strategy Pattern wird großenteils verwendet um Verhaltensmuster darzustellen. Nehmen wir an, mehrere Klassen sollen die gleichen Funktionen beinhalten, aber sie sollen auf eine andere Art und Weise durchgeführt werden. Nehmen wir ein einfaches Beispiel: In einem Spiel gibt es verschiedene Charaktere. Nun soll jeder Charakter eine spezielle Tätigkeit ausführen, welche für ihn ausschlaggebend ist. Die Tätigkeit soll bei jedem gleich aufgerufen werden können, unabhängig von dem Charakter oder der Tätigkeit. Dennoch können die Tätigkeiten pro Person unterschiedlich, aber auch gleich sein. Hierzu eignet sich das Strategy Pattern.

UML



Funktionsweise

Bleiben wir gleich bei dem oben genannten Beispiel. Demnach wäre ein spezieller Charakter (z.B. Schmied) im oben gezeigten UML Diagramm ein ConcreteComponent welcher von einer Klasse (in dem Fall z.B Charakter) erbt. Diese Klasse beinhaltet eine Referenz auf die Tätigkeit, welche dieser ausführt. In einer Methode in der spezifischen Charakter-Klasse kann nun die Tätigkeit der Person über eine Methode im Attribut (Referenz auf Tätigkeit) aufgerufen werden, welche bei jeder Tätigkeit gleich benannt ist. Um also die Tätigkeit zu ändern, reicht es aus die Referenz zu ändern. Die Tätigkeiten allgemein, werden durch die Abstraction dargestellt, welche ein Interface ist, um sicherzustellen, dass die Methode, die vom Charakter aufgerufen wird, vorhanden ist. Die ConcreteAbstraction hingegen stellt eine spezielle Tätigkeit, wie z.B. das Tischlern dar. Diese müssen das Interface implementiert haben, um überhaupt als Tätigkeit erkannt zu werden.

Vorteile und Nachteile

Vorteile

- Algorithmen sind austauschbar
- Algorithmen sind unabhängig von Klassen

Nachteile

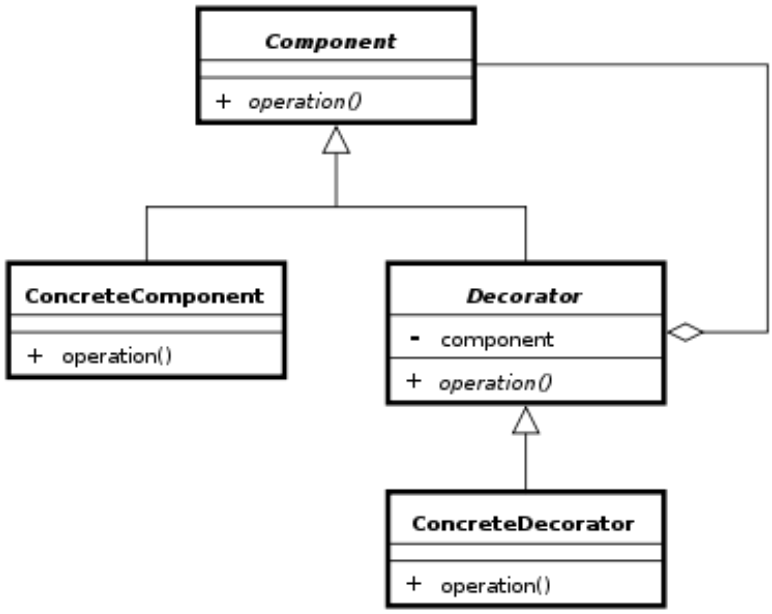
- Größerer Kommunikationsaufwand zwischen Strategie und Kontext
- Größere Anzahl Objekte
- Klienten müssen die möglichen Strategien kennen

Decorator

Warum Decorator?

Theoretisch ist das Decorator Pattern unendlich oft erweiterbar. Jede Erweiterung kann unterschiedlich oft und in verschiedenen Reihenfolgen angewandt werden. Sie muss nur von der vorgegebenen Decorator Klasse (abstract) erben.

UML



Funktionsweise

An der Spitze steht bei dem Decorator Design Pattern immer ein Interface, oder eine „abstract class“ um zu gewährleisten, dass alle wichtigen Methoden vorhanden sind, damit das erweitern(umhüllen) funktioniert, ohne Anpassungen am restlichen Code vorzunehmen, indem alle Decorator gleich aufgebaut sind. Der „ConcreteComponent“ wird in den meisten Fällen dazu benutzt, das durch Decorator veränderte Attribut an andere Teile derselben, oder anderer Software, weiterzugeben. Er könnte es natürlich auch zu Weiterverarbeitung an einen Server senden ausprinten, oder aus der anderen Richtung gesehen von einem Socket, aus Files, aus einer GUI, oder aus der Konsole auslesen. Die Decorator müssen alle noch eine Variable beinhalten, in welcher das zu dekorierende Element eingeschlossen wird. Da die einzige Bedingung, welche dieses Element zu erfüllen hat, die ist, von dem im Beispiel gezeigten Component zu erben(„oder ihn zu implementieren“), kann auch ein Decorator den Anderen umschließen, was wiederum einer theoretisch endlose Erweiterung erlaubt.

Vor und Nachteile

Vorteile

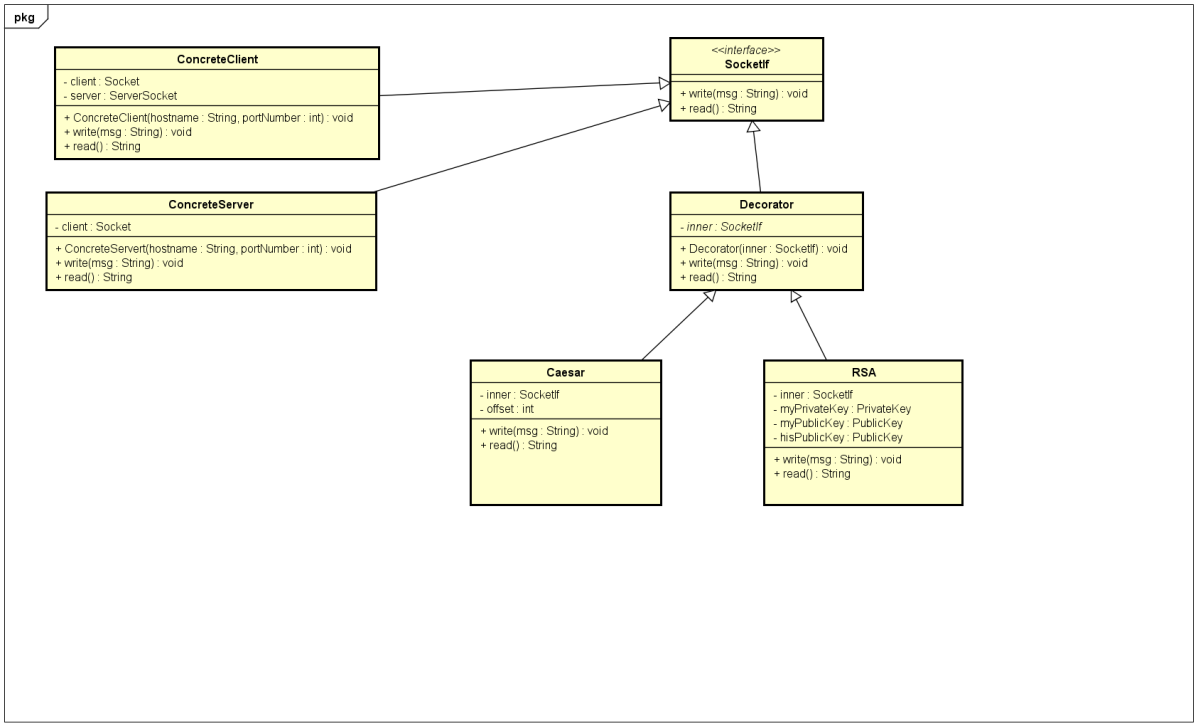
- Beliebige Anzahl von Erweiterungen
- Decorator können Ergebnisse verändern (z.B. Codieren)
- Transparenz
- Decorator und konkrete Klassen sind vom selben Typ

Nachteile

- Unübersichtlich
- Pro Decorator eine Klasse

A06 Aufgabe

UML Diagramm



Funktionsweise

Wie bei dem Decorator Pattern üblich, steht an der Spitze aller dekorierbaren und dekorierenden Klassen ein Interface, welches sicherstellt, dass die erforderlichen Methoden (hier read() und write()) vorhanden sind. Der Decorator(als abstract class) gibt vor, dass auch noch eine Instanz des SocketIf Interfaces vorhanden sein muss, da bei einer dekorierenden Klasse immer ein „inneres“ Element vorhanden sein muss. Die Decorator können sich somit beliebig oft „wrappen“(umwickeln) und es werden immer nur die read() und write() Methoden verwendet. Bei dem Server und Client werden, im Gegensatz zu den Dekoratoren die Inhalte nicht einfach an die inneren Elemente weitegegeben, sondern direkt über den Socket an die jeweils andere Seite(Server→Client, Client→Server) gesandt.