Wrapper & More

Dürfen wir es ihnen einpacken?

Clean Code Prinzipien (Auszug)

- KISS → Keep It Simple and Stupid
- YAGNI → You Ain't Gonna Need It
- Don't optimize prematurely (applies to performance, structure, and design pattern)
  - The first rule of optimization is: Don't do it.
  - The second rule of optimization (for experts only) is: Don't do it yet.

Aufgabe

Logging implementieren

Projekt “PdfTools”

Die Strategies sollen die Parameter loggen
Do()-Methode erweitern

Agenda

01 Decorator Pattern
...und ein bisschen
Proxy Pattern

02 Adapter Pattern

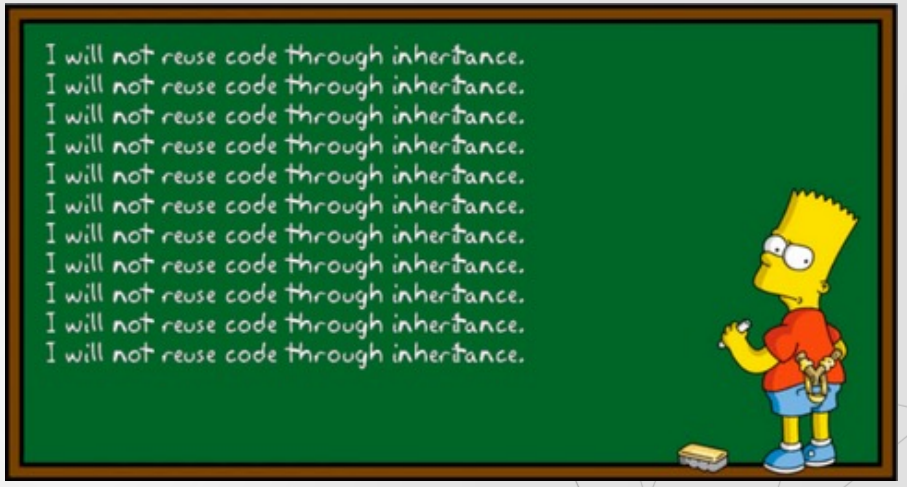
03 ~~Facade Pattern~~

01

Decorator Pattern

Favour Composition over Inheritance

- Komposition statt Vererbung
- Entkoppeln der Klassen
- Verändern des Verhaltens zur Laufzeit



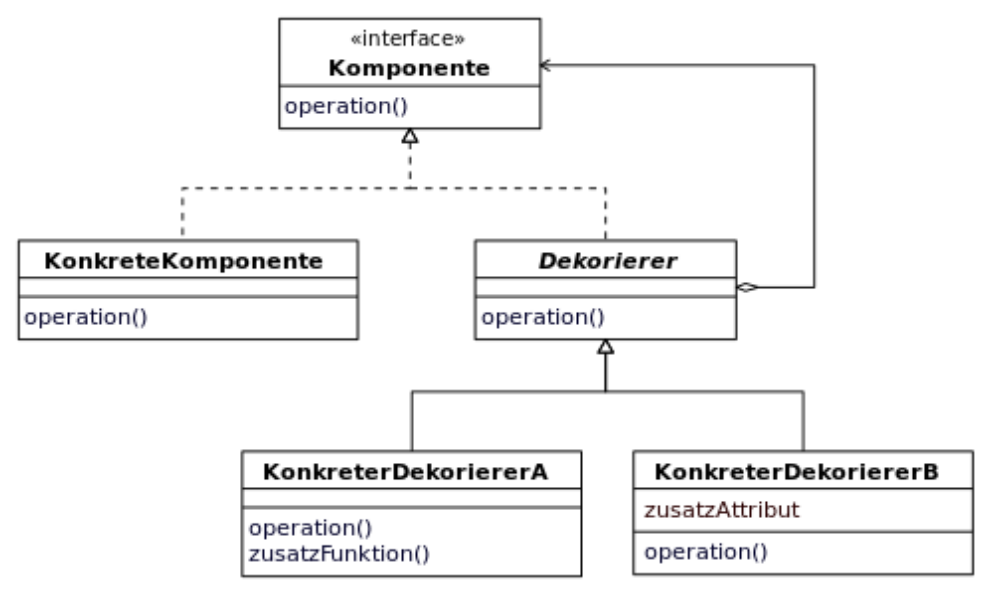
- Vererbung: B erbt von A
- Komposition: B hat ein Feld mit A

Decorator Pattern

- Flexible Alternative zur Unterklassenbildung
- Klasse um zusätzliche Funktionalitäten zu erweitern
- Zum Beispiel: Logging, abgerundete Fenster

- Dekorierer hat (indirekt) die gleiche Schnittstelle wie die zu dekorierende Klasse
- Transparent aus Sicht des Aufrufenden

Say it with UML



Aufgabe

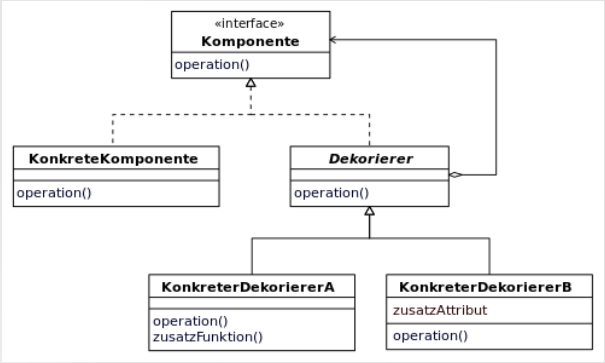
Logging implementieren

Projekt “PdfTools”

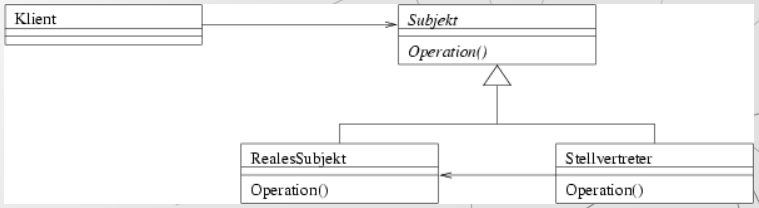
Die Commands sollen die Parameter loggen

LoggingCommandDecorator: ICommand
Konstruktor mit ICommand als Parameter
New LoggingCommandDecorator(new AddCodeCommand)

Proxy vs. Decorator Pattern



- Decorator hat keinen (fachlichen) Bezug zum Decoratee. Erbt vom Interface.
    a. LoggingDecorator
- Proxy erweitert die fachliche Funktionalität. Erbt i.d.R. von der Klasse.
    a. CachingWebProxy
    b. Konto → KontoMitPasswort



02

Adapter Pattern

Extract 3rd Party Dependency

- Projekt für die Contracts anlegen (Schnittstellen) → “Layer of Abstraction”
- Projekt für die konkrete Implementierung anlegen mit Anhängigkeit zu 3rd Party Deps.
    a. “public API” sind nur Schnittstellen
    b. Factory erzeugt ggf. die konkrete Implementierung
- Ausgangsprojekt hat die Abhängigkeit nur indirekt (transitiv)
- Contracts sind die gemeinsame Referenz

Adapter Pattern

- Auch “Hüllenklasse” oder “Wrapper”
- Übersetzung Schnittstelle (intern) nach Schnittstelle/Klasse (extern)
- Kompatibilität von heterogenen Bibliotheken

- Der Adapter implementiert die projektinterne Schnittstelle
- Der Adapter konvertiert die Methodenaufrufe auf die externe Schnittstelle

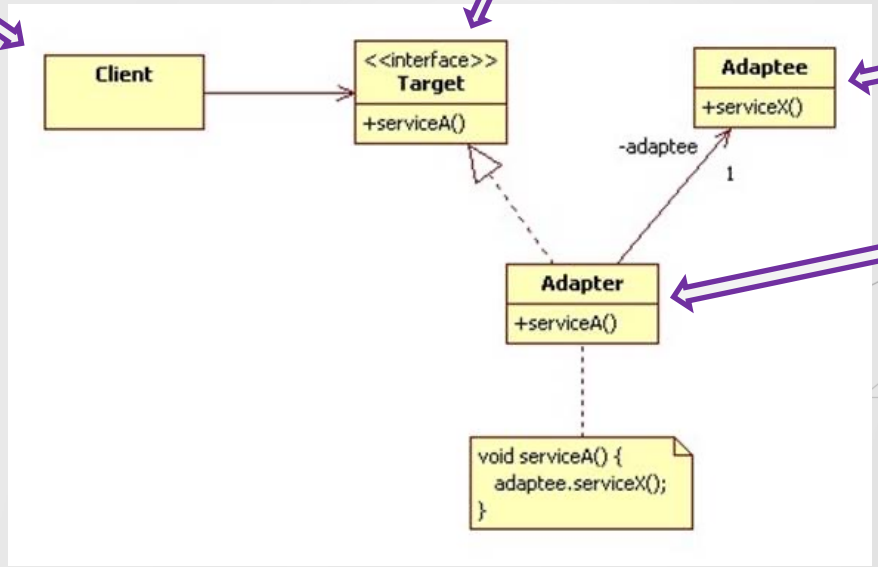
UML Diagramm

PdfTools

PdfTools.Contracts

NLog

PdfTools.Logging.NLog



Aufgabe

Abstraktion definieren
Nlog Log Unabhaengigkeit

Projekt: „PdfTools“

IPdfLogger (Debug(), Error())
NLogLoggerAdapter
Ggf. Projekte für Contracts und Implementierung
Ggf. Factory NLogLoggerFactory erstellen