Wrapper & More

Dürfen wir es ihnen einpacken?

Agenda

01 ~~Decorator Pattern~~
~~...und ein bisschen~~
~~Proxy Pattern~~

02 ~~Adapter Pattern~~

03 Facade Pattern

04 State Pattern

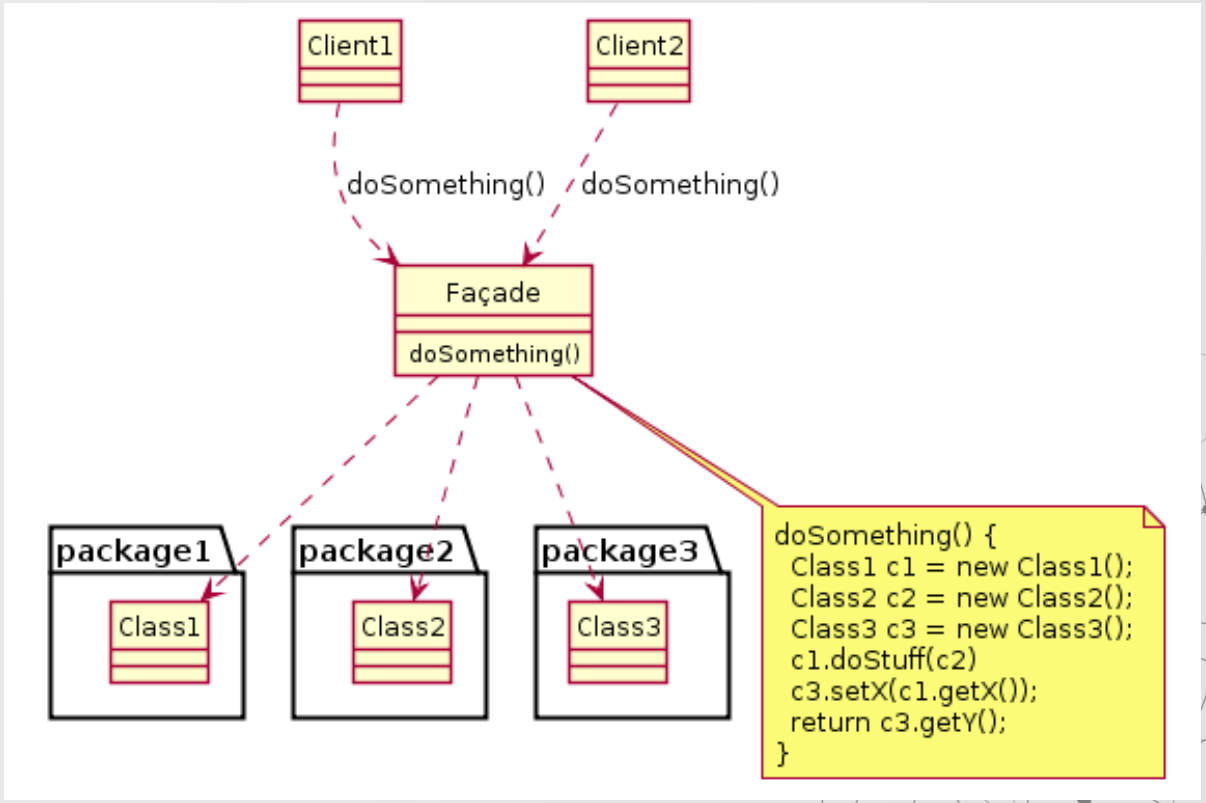
03

Facade Pattern

Facade Pattern

- Vereinfachte Schnittstelle zu einem Subsystem
- Verbirgt die Details des Subsystems
- Verbergen der technisch orientierten Klassen des Subsystems
- Bereitstellen der verwendeten Funktionen

Facade Pattern



Aufgabe

Extract HttpClient

Projekt: “PdfTools”

Facade für den HttpClient erstellen
HttpClientFacade für Einfachheit

IHttpClient für Testbarkeit

Resuemee Wrapper

Wrapper

- Erweitern der Funktionalität (Proxy, Decorator)
- Verbergen von Funktionalität (Façade)
- Anbinden von Funktionalität (Adapter)

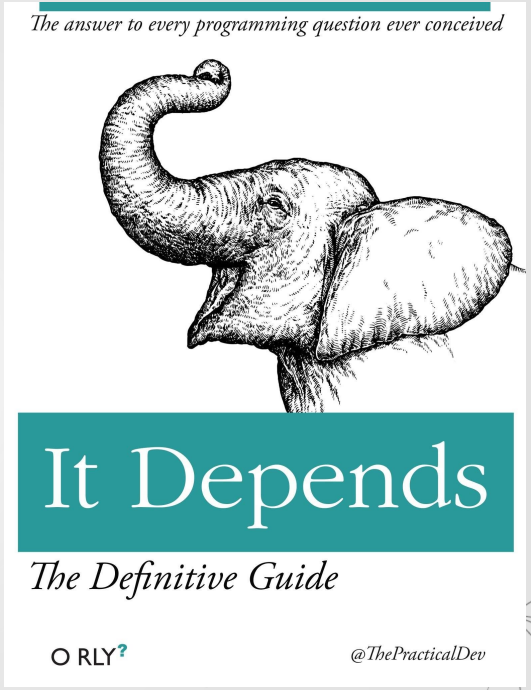
- Separation of Concerns
- Testbarkeit
- Dynamisches Verhalten der Applikation

Moegliche Loesungen zum Dekorieren

Wie fügt man (optionales) “Logging” zu einer Klasse hinzu

- Decorator Pattern
- Null-Object Pattern
- Vererbung
- (Aspekt Orientierte Programmierung)

- Und sicherlich viele mehr…



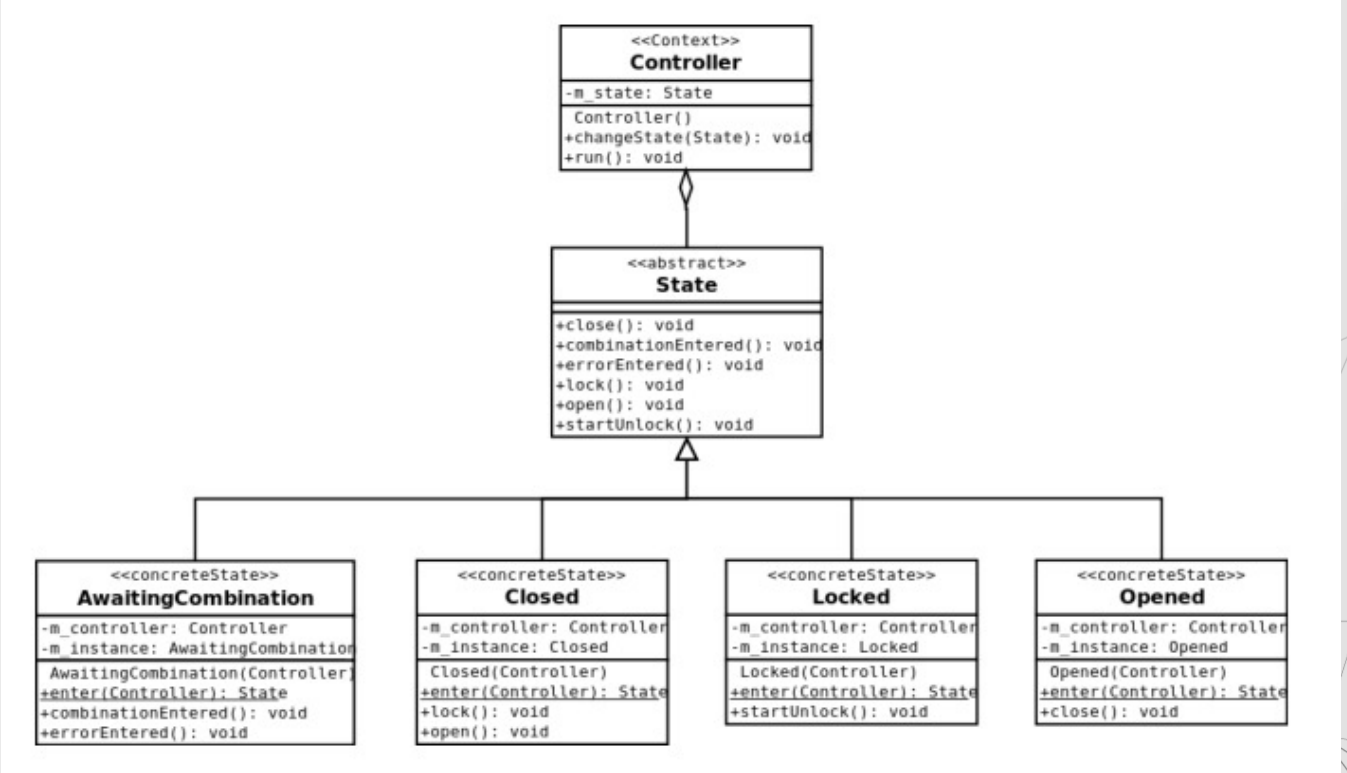
04

State Pattern

State Pattern

- Verhalten einer Klasse ändern, je nach Zustand
- Jeder Zustand hat eine eigene Klasse (“Zustands-Strategie)
- Zustandsänderungen ändern die Instanz/Klasse für den State
- Der Zustand (Werte) werden übergeben (Dispatch “this”)

UML Diagram State Pattern



Aufgabe/Demo

State fuer das Team

Projekt “Person”
Klasse “Team”

Team-Methoden: Urlaub(), Mache(), BeendeArbeit()
Team Property “State : Istate”
Zustände (IState): ImUrlaub, Beschäftigt, Frei

Beispiel: ImUrlaub.Mache() {_team.State = new Beschäftigt()}