|  |
| --- |
| SpitApp |
| State Pattern |
| Team Grün – Swen Lanthemann, Roger Jaggi, Pascal von Ow, Pascal Bohni |

21.3.2013 Version 2

Inhalt

[2 Einleitung 2](#_Toc356552072)

[2.1 Erklärung 2](#_Toc356552073)

[2.2 Anwendungsgebiet 2](#_Toc356552074)

[3 Bewertung 3](#_Toc356552075)

[3.1 Vorteile 3](#_Toc356552076)

[3.2 Nachteile 3](#_Toc356552077)

[4 Quellen 4](#_Toc356552078)

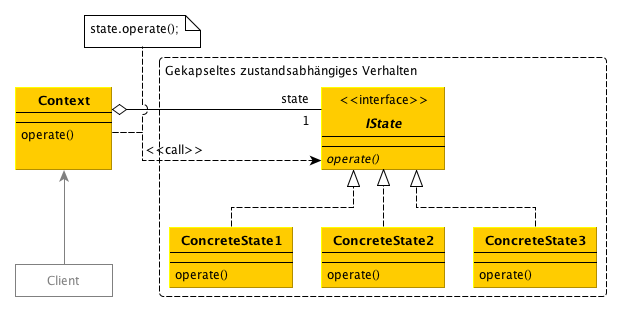
# Einleitung

## Erklärung

Das Statepattern gehört zu den Verhaltensorientierten (behavioral) Patterns.

Das zustandsabhängige Verhalten des Objekts wird in separate Klassen ausgelagert, wobei für jeden möglichen Zustand eine eigene Klasse eingeführt wird, die das Verhalten des Objekts in diesem Zustand definiert. Damit der Kontext die separaten Zustandsklassen einheitlich behandeln kann, wird eine gemeinsame Abstrahierung dieser Klassen definiert.

Bei einem Zustandsübergang tauscht der Kontext das von ihm verwendete Zustandsobjekt aus.



## Anwendungsgebiet

Prinzipiell kann jedes zustandsabhängige Verhalten durch dieses Entwurfsmuster abgebildet werden. Beispielsweise wird es für die Verwaltung von Sessions oder von Ein- und Ausgabeströmen, bei zustand behafteten Bedienelementen einer grafischen Benutzeroberfläche oder bei Parkautomaten verwendet.

Ein Zeichen dafür dass, das Statepattern angewendet werden könnte ist wenn man irgendwo eine switch Anweisung im Code sieht oder anwenden möchte.

In Spitapp☺ wenden wir das Pattern beim holen der TerminEinträge an.

# Bewertung

## Vorteile

* Das System kann bei Bedarf leicht mit neuen States ergänzt werden.
* Verständlichkeit dank Status
* Besser als IF/ELSE Konstrukte und dadurch auch weniger Fehleranfällig

## Nachteile

* Viele Klassen (bei vielen States…)

# Quellen

<http://de.wikipedia.org/wiki/Zustand_(Entwurfsmuster)>

<http://www.philipphauer.de/study/se/design-pattern/state.php>