Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

LE 09 – Entwurf mit Design Pattern II
Zusammenfassung

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Lernziele LE 09 – Entwurf mit Design Patterns II

- Sie sind in der Lage:
  - Den Aufbau der folgenden Design Patterns zu erklären und sie anzuwenden:
    - Decorator
    - Observer
    - Strategy
    - Composite
    - State
    - Visitor
    - Facade

Agenda

1. **Repetition Aufbau von Design Patterns**
2. Design Patterns
3. Wrap-up und Ausblick

Repetition Aufbau Design Patterns

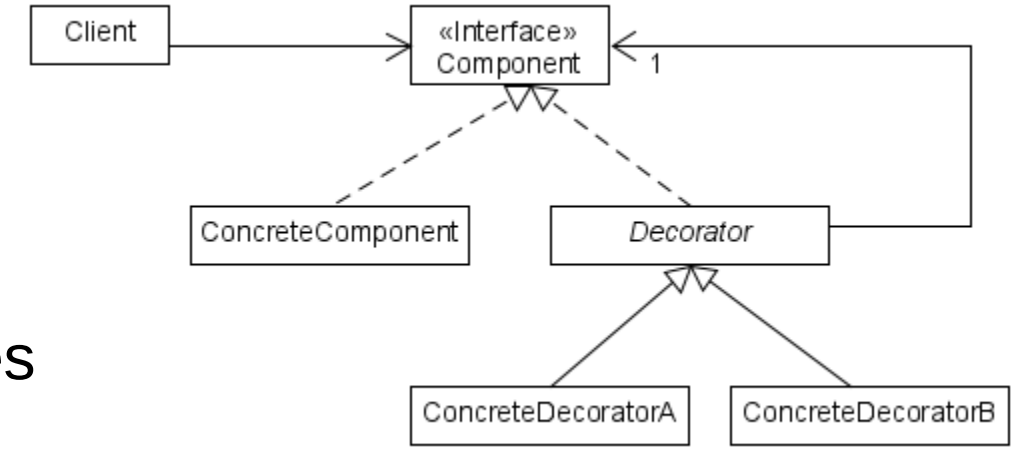
- Beschreibungsschema:
  - Name
  - Beschreibung Problem
  - Beschreibung Lösung
  - Hinweise für Anwendung
  - Beispiele
- GRASP: Design Prinzipien
- GoF: Ausgefeiltere Spezialfälle von GRASP

Agenda

1. Repetition Aufbau von Design Patterns
2. **Design Patterns**
3. Wrap-up und Ausblick

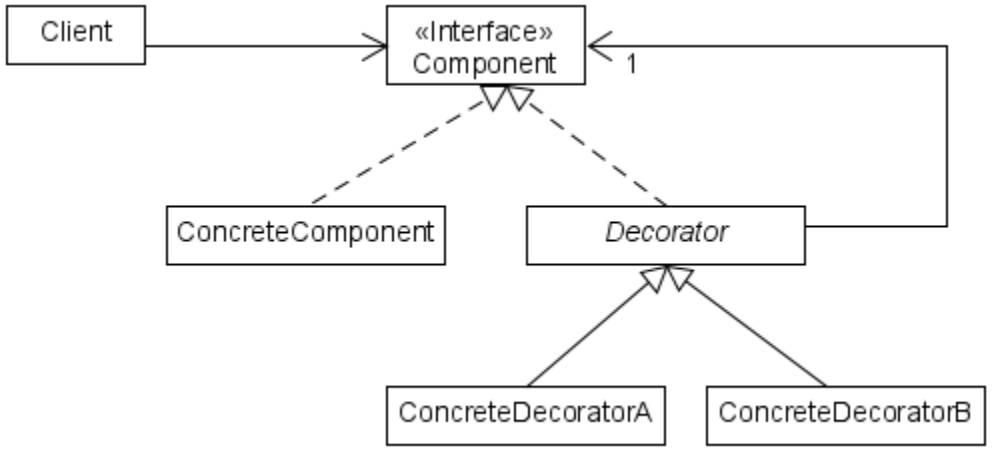
Decorator: Problem und Lösung

- Problem:
  - Ein Objekt (nicht eine ganze Klasse) soll mit zusätzlichen Verantwortlichkeiten versehen werden.
- Lösung
  - Ein Decorator, der dieselbe Schnittstelle hat wie das ursprüngliche Objekt, wird vor dieses geschaltet. Der Decorator kann nun jeden Methodenaufruf entweder selber bearbeiten, ihn an das ursprüngliche Objekt weiterleiten oder eine Mischung aus beidem machen.



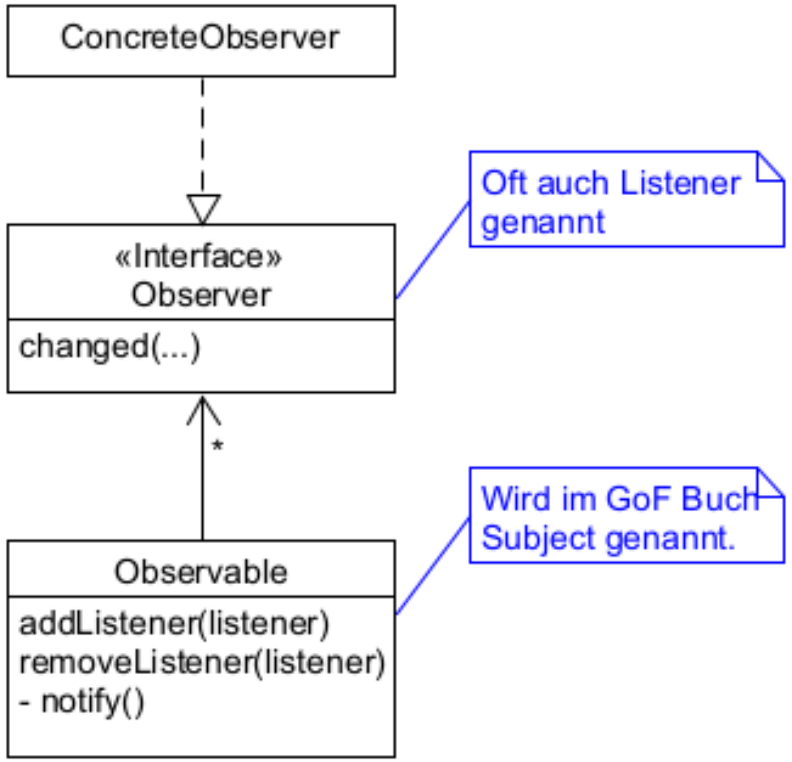
Decorator: Hinweise

- Hinweise
  - Strukturell identisch mit dem Proxy Design Pattern, hat aber eine andere Absicht.
  - Eigentlich identisch mit dem «Composite» Design Pattern, wenn die Anzahl Elemente 1 ist, hat aber natürlich auch eine andere Absicht.



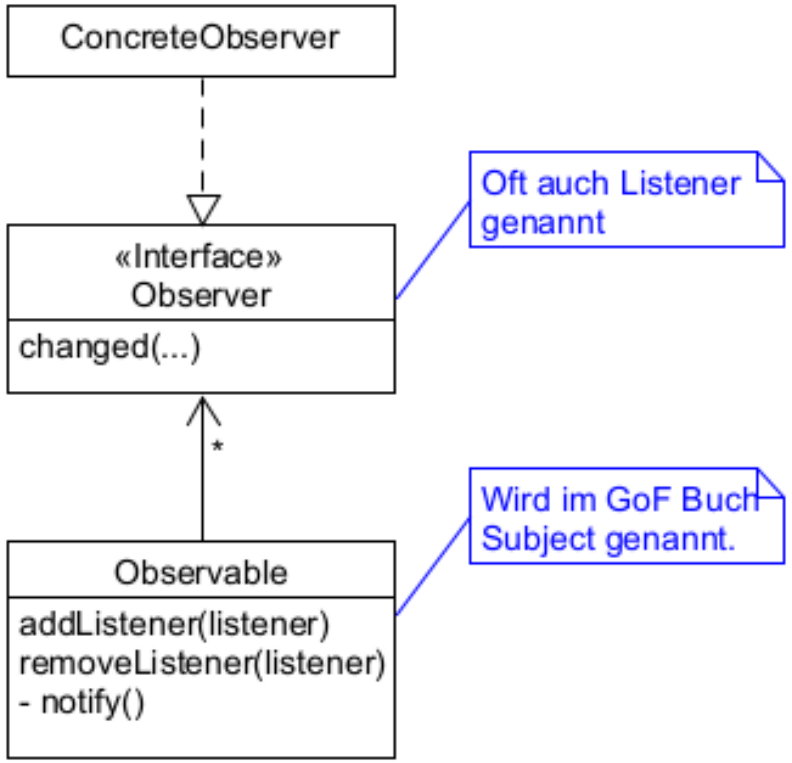
Observer: Problem und Lösung

- Problem:
  - Ein Objekt soll ein anderes Objekt benachrichtigen, ohne dass es den genauen Typ des Empfängers kennt.
- Lösung
  - Ein Interface wird definiert, das nur dazu dient, ein Objekt über eine Änderung zu informieren. Dieses Interface wird vom «Observer» implementiert. Das «Observable» Objekt benachrichtigt alle registrierten «Observer» über eine Änderung.



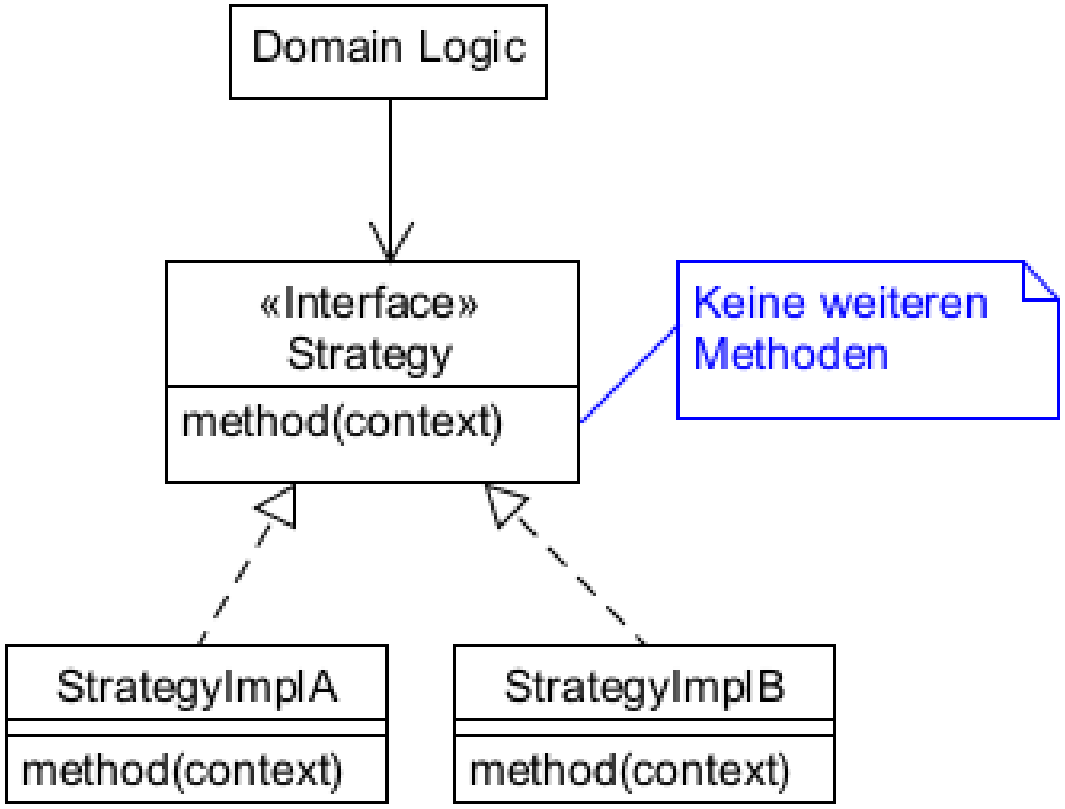
Observer: Hinweise

- Hinweise
  - Oft wird dieses Pattern auch «Publish-Subscribe» genannt.
  - Observable kennt nur Observer, aber nicht den wahren Typ ConcreteObserver.
  - 2 Phasen : Zuerst die Registrierung der Observer, dann die Benachrichtigungen durch das Observable.



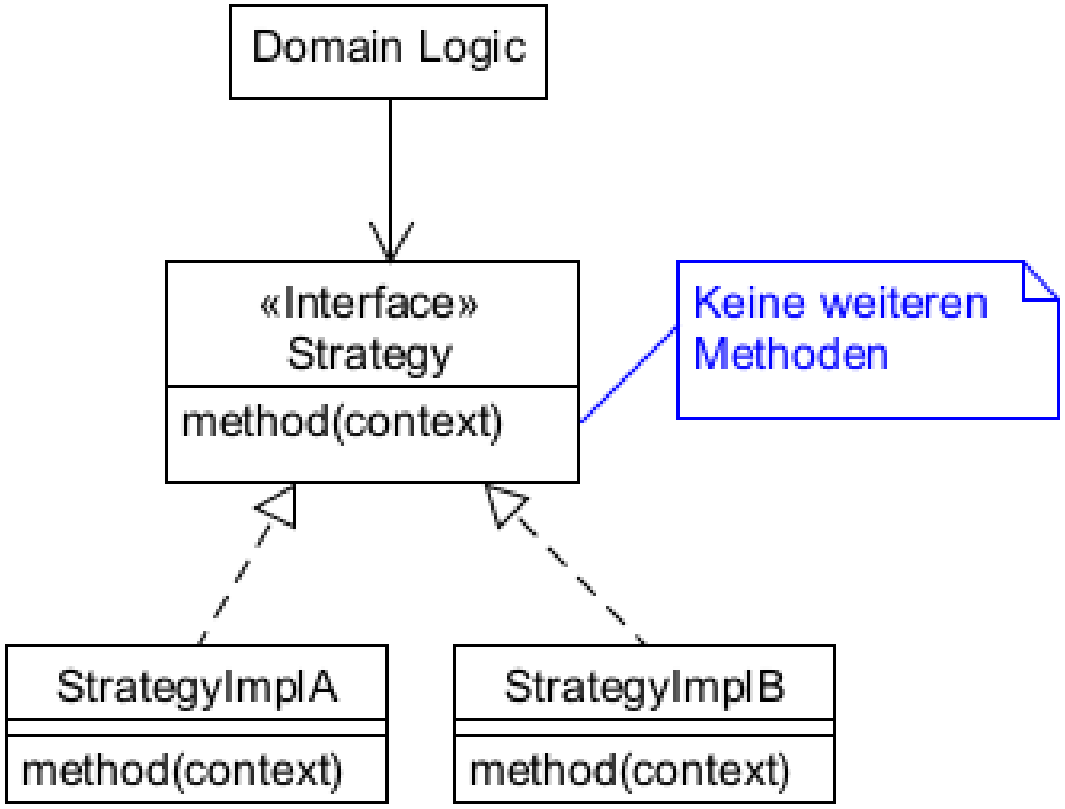
Strategy: Problem und Lösung

- Problem:
  - Ein Algorithmus soll einfach austauschbar sein.
- Lösung
  - Den Algorithmus in eine eigene Klasse verschieben, die nur eine Methode mit diesem Algorithmus hat.
  - Ein Interface für diese Klasse definieren, das von alternativen Algorithmen implementiert werden muss.



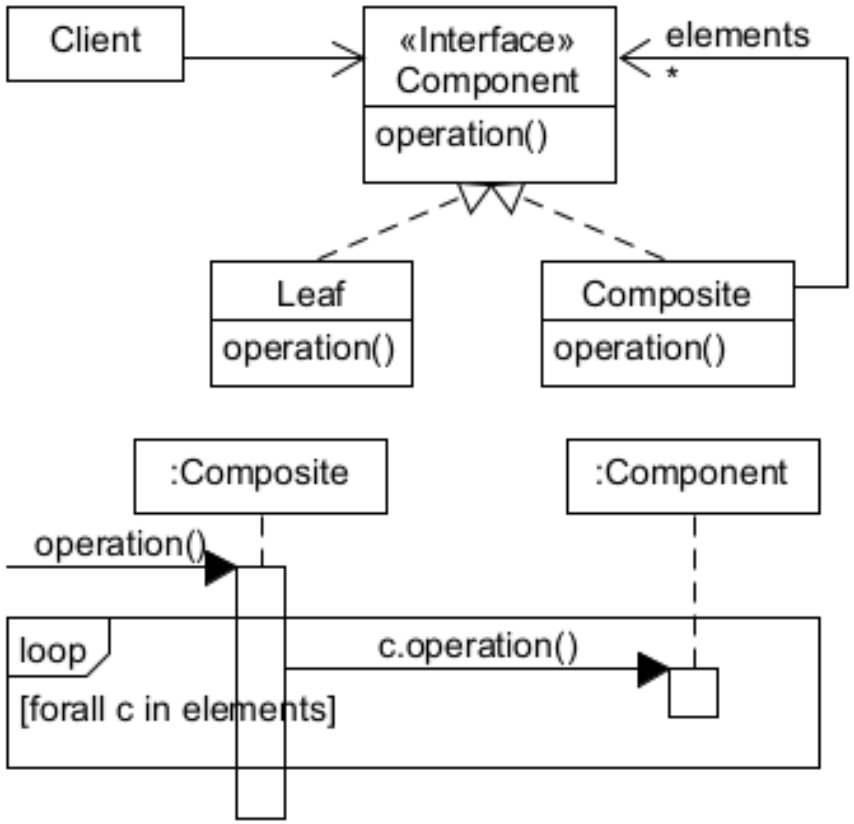
Strategy: Hinweise

- Hinweise
  - Als Motivation für die einfache Austauschbarkeit können technische wie auch fachspezifische Gründe vorhanden sein.
  - Das Interface hat nur eine einzige Methode. Als Parameter müssen alle Daten übergeben werden, die der Algorithmus benötigt. Dieser Parameter wird üblicherweise «context» benannt.



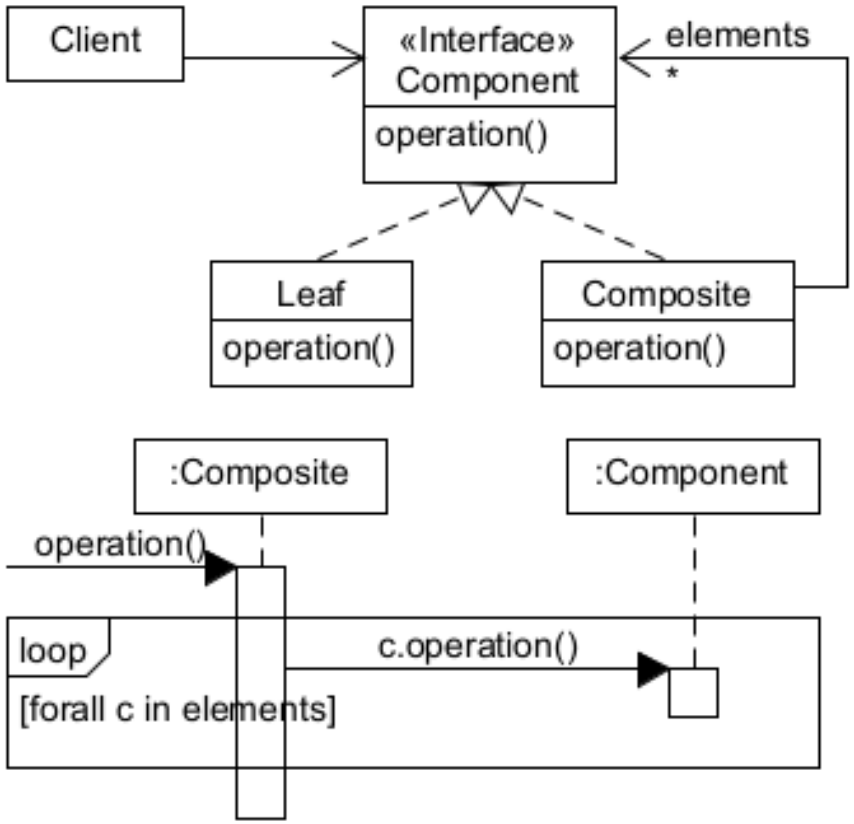
Composite: Problem und Lösung

- Problem:
  - Eine Menge von Objekten haben dasselbe Interface und müssen für viele Verantwortlichkeiten als Gesamtheit betrachtet werden.
- Lösung
  - Sie definieren ein Composite, das ebenfalls dasselbe Interface implementiert und Methoden an die darin enthaltenen Objekte weiterleitet.



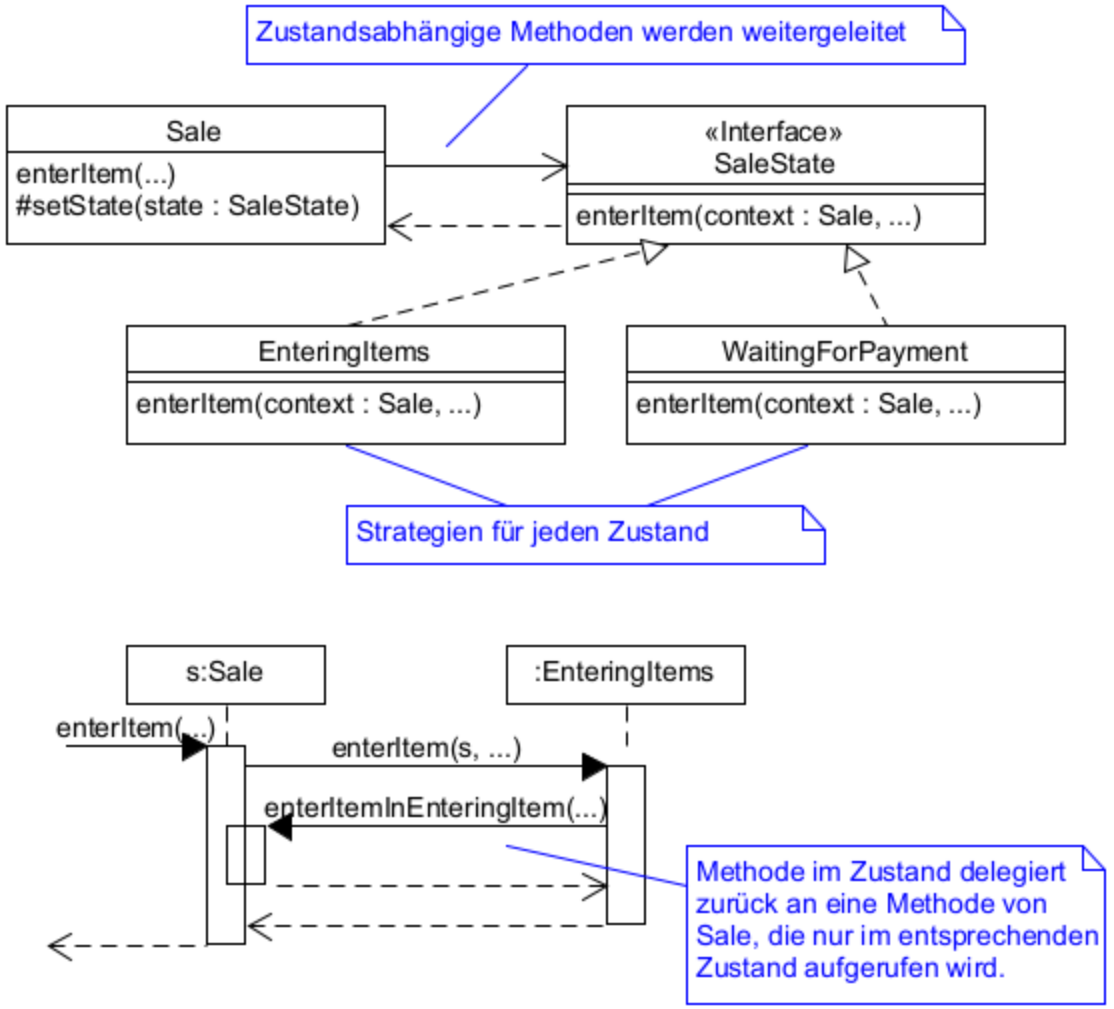
Composite: Hinweise

- Hinweise
  - Oft ist die hierarchische Struktur vom Fachgebiet her gegeben.
  - Nicht alle Methoden delegieren einfach auf die enthaltenen Elemente. Vor- und Nachbearbeitung ist üblich, und gewisse Methoden müssen ganz anders implementiert werden.



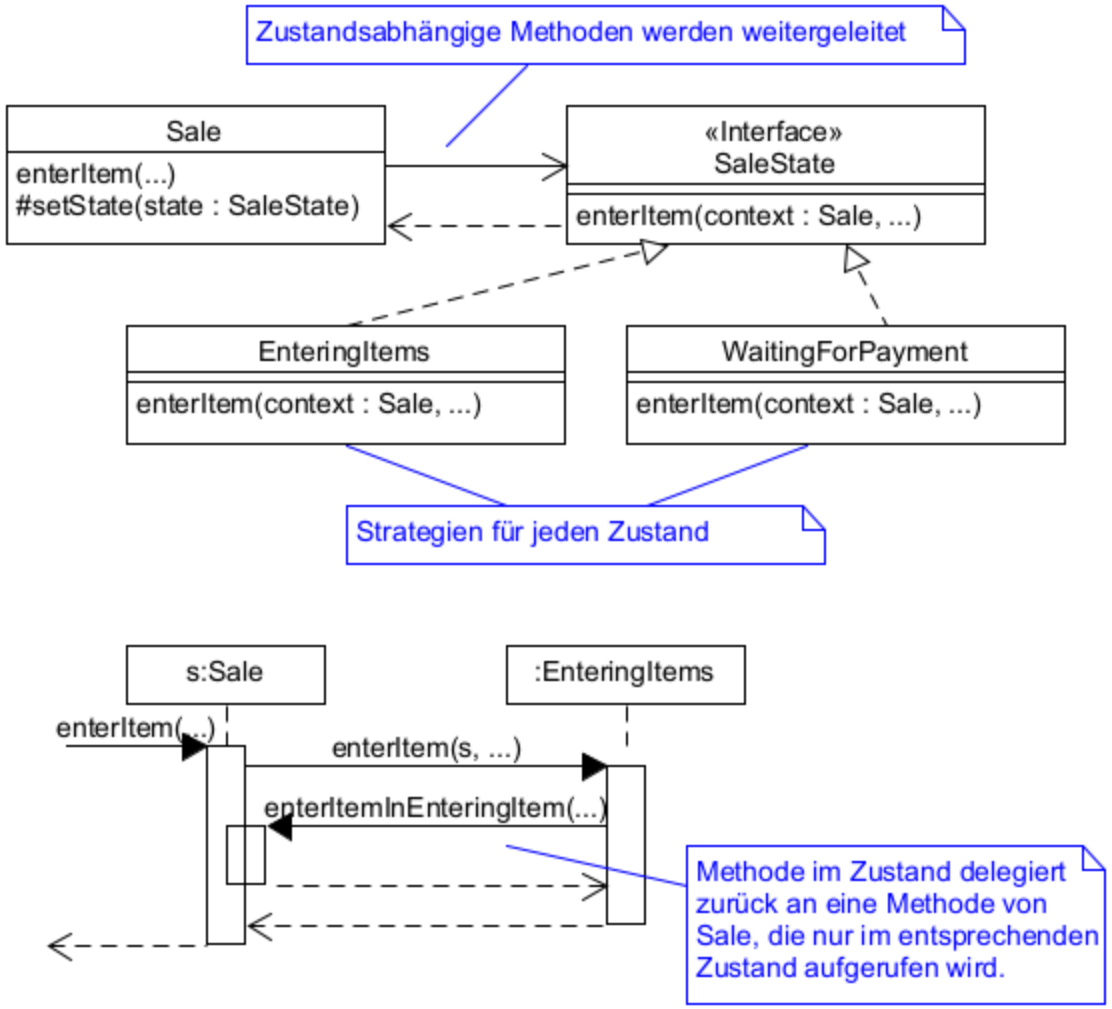
State: Problem und Lösung

- Problem:
  - Das Verhalten eines Objekts ist abhängig von seinem inneren Zustand.
- Lösung
  - Das Objekt hat ein darin enthaltenes Zustandsobjekt.
  - Alle Methoden, deren Verhalten vom Zustand abhängig sind, werden über das Zustandsobjekt geführt.



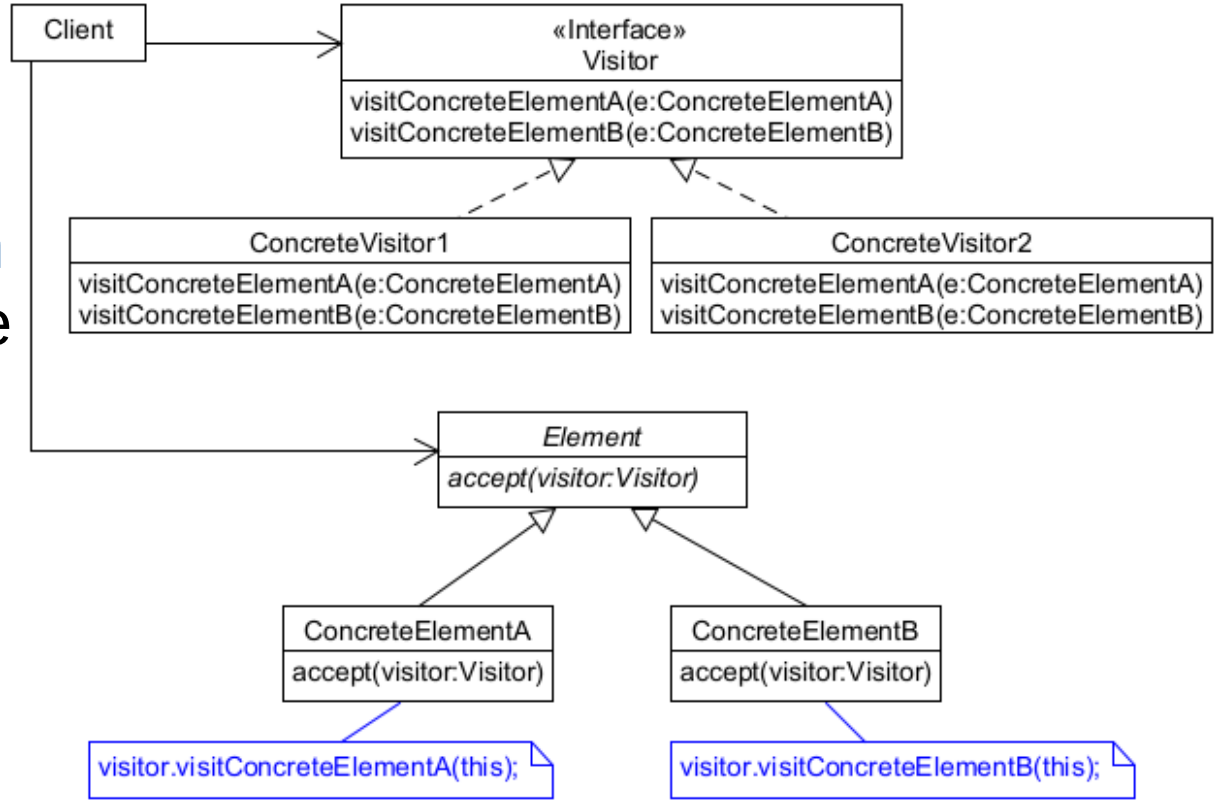
State: Hinweise

- Hinweise
  - Die Zustands-Klassen implementieren das Zustand-Interface.
  - Die Zustands-Objekte sind nichts anderes als Strategy Objekte und können Singletons sein.
  - Das Zustandsobjekt hat entweder direkt den Code (als innere Klasse) oder delegiert an eine Methode des Objekts weiter.



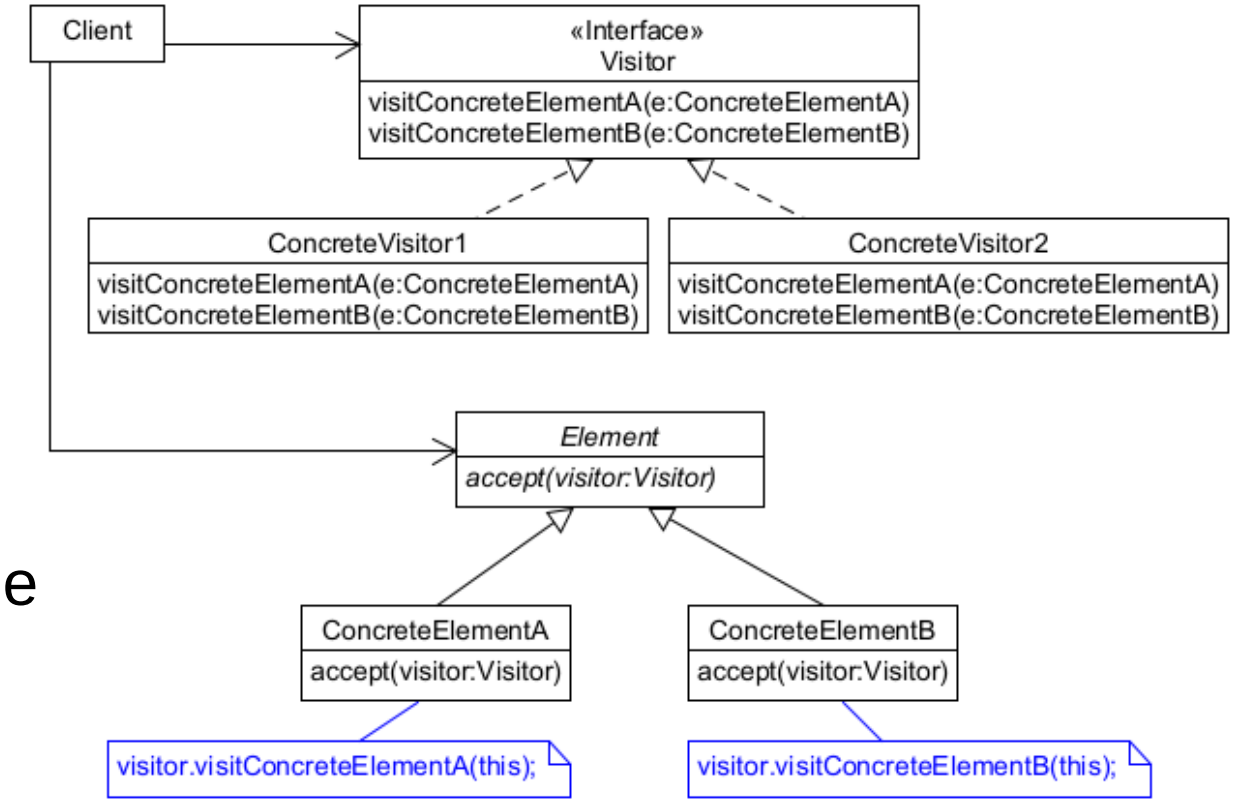
Visitor: Problem und Lösung

- Problem:
  - Eine Klassenhierarchie soll um (weniger wichtige) Verantwortlichkeiten erweitert werden, ohne dass viele neue Methoden hinzukommen.
- Lösung
  - Die Klassenhierarchie wird mit einer Visitor-Infrastruktur erweitert. Alle weiteren neuen Verantwortlichkeiten werden dann mit spezifischen Visitor-Klassen realisiert.



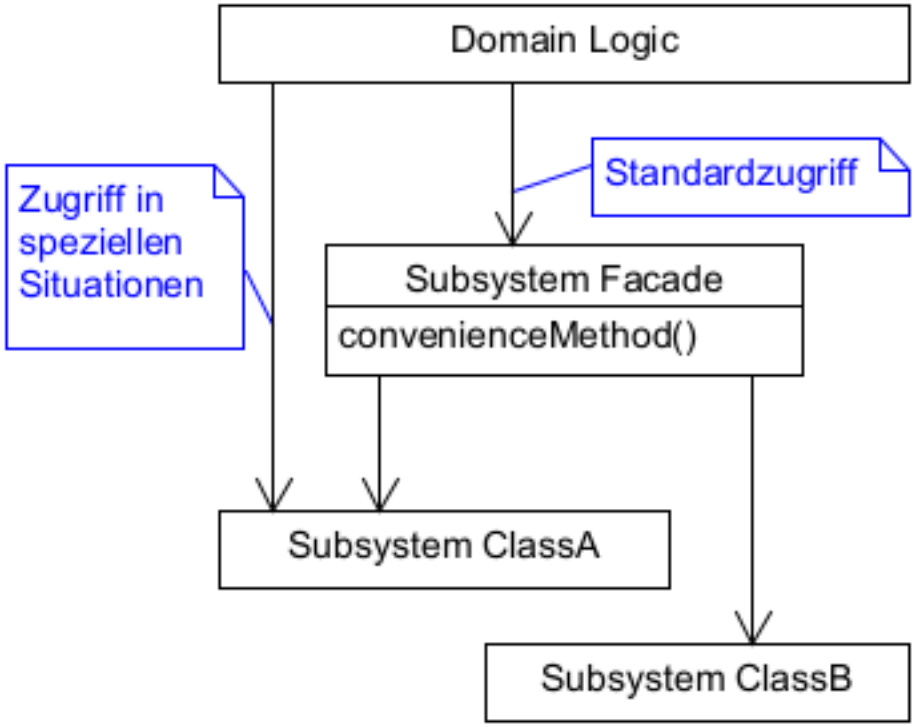
Visitor: Hinweise

- Hinweise
  - Widerspruch zum Information Expert. Daher wichtige Methoden weiterhin direkt der Klasse hinzufügen.
  - Oft werden Auswertungen an Visitor-Klassen delegiert.
  - Bei einer mehrstufigen Objekthierarchie stellt sich die Frage, wer die darin enthaltenen Elemente aufruft. Siehe Beispiel Schachprogramm.



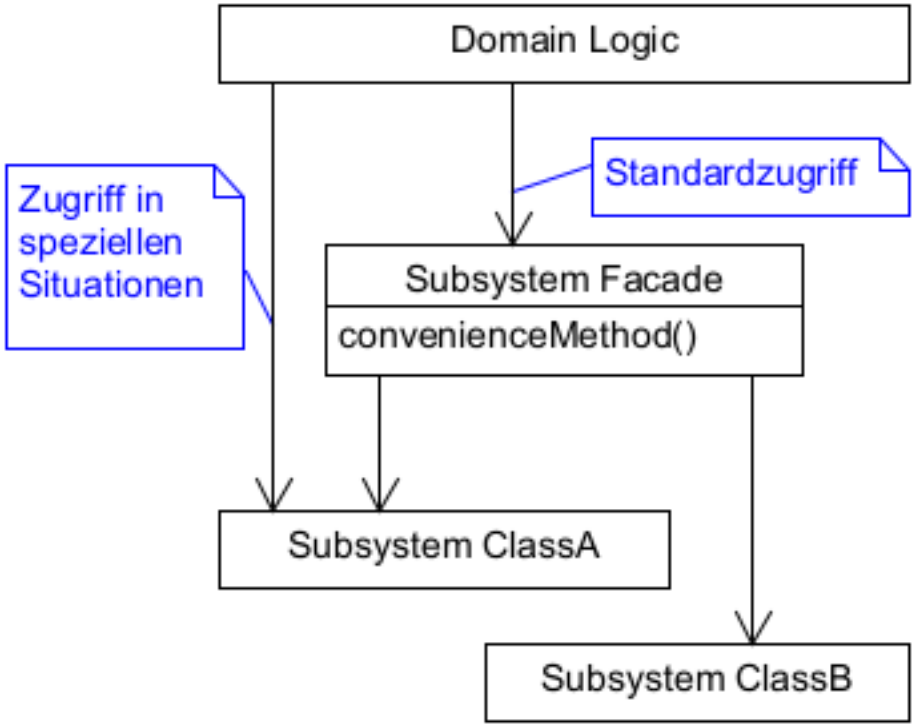
Facade: Problem und Lösung

- Problem:
  - Sie setzen ein ziemlich kompliziertes Subsystem mit vielen Klassen ein. Wie können Sie seine Verwendung so vereinfachen, dass alle Team-Mitglieder es korrekt und einfach verwenden können?
- Lösung
  - Eine Facade (Fassade) Klasse wird definiert, welche eine vereinfachte Schnittstelle zum Subsystem anbietet und die meisten Anwendungen abdeckt.



Facade: Hinweise

- Hinweise
  - Eine Facade kapselt, im Gegensatz zum Adapter, ein Subsystem nicht vollständig ab. Es ist erlaubt, dass die Methoden der Facade Parameter und Rückgabewerte haben, die Bezug auf das Subsystem nehmen.
  - Wird oft vom Ersteller eines Frameworks entwickelt.



Agenda

1. Repetition Aufbau von Design Patterns
2. Design Patterns
3. **Wrap-up und Ausblick**

Wrap-up

- Ein Decorator erweitert die Funktionalität eines Objekts (im Gegensatz zu Vererbung)
- Ein Observer beobachtet das Observable. Da der Observer dem Observable nur als Interface bekannt ist, wird Low Coupling unterstützt.
- Eine Strategy ist ein Klasse, die genau einen Algorithmus enthält. Über Polymorphismus kann dann die Strategy einfach ausgetauscht werden.
- Ein Composite beinhaltet Objekte, die dasselbe Interface wie das Composite implementieren. Viele Methoden werden dann auf diese Objekte weitergeleitet.
- Zustandsabhängiges Verhalten wird über ein State Objekt geleitet.
- Ein Visitor besucht Objekte, die dann die richtige Methode auf dem Visitor aufrufen.
- Ein Facade bietet für ein Teilsystem eine vereinfachte Benutzung an.

Ausblick

- In der nächsten Lerneinheit werden wir:
  - Einen Quiz zum Design durchführen.
  - Den Prozess des Refactorings genauer anschauen.

Quellenverzeichnis

[1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005

[2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012

[3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018

[4] Gamma, E et al.: Design Patterns: Elements of Reusable Object-Oriented Software Addison Wesley Longman, 1995

[5] McDonald, J: DZone Refcardz: Design Patterns, www.dzone.com, 2008