Übung 4:

# Diskussion Open/Close Verstoss in der Restaurantsimulation

## Beschreiben Sie diese Situation und den Grund für den Verstoss

Gemäss den Open/Close Principle von SOLID sollten Klassen offen für Erweiterungen und geschlossen für Modifikationen sein. Mit den neuen Anforderungen, dass ein Restaurantleiter das Geschehen im Restaurant verfolgt und Personal an Essensausgaben abstellt/entfernt und Kassen öffnet/schliesst, sollte es klar sein, dass wir unsere bestehenden Klassen erweitern müssen.

Leider gestaltet sich das nicht so einfach, da mit den neuen Anforderungen die Klasse Counter plötzlich zwei Ziele erfüllen muss:

1. Für die Essensausgabe an den Theken kann Personal abgestellt und wieder entfernt werden. Die Anzahl der Queues, nämlich eine, verändert sich dabei nie, sondern nur die Abarbeitungsgeschwindigkeit (Diese muss jetzt plötzlich änderbar sein)
2. Für die Kassen kann Personal abgestellt und wieder entfernt werden. Dabei kann eine Kasse geöffnet werden und neue Personen annehmen (Eine Kasse muss immer geöffnet sein). Wird die Klasse geschlossen, müssen die anstehenden Personen auf eine andere Kasse verteilt werden (Die Kassen müssen sich also gegenseitig kennen). Die Abarbeitungsgeschwindigkeit innerhalb der Queue ändert sich nie

Diese zwei Ziele führen zu einer unterschiedlichen [konfliktbehafteten] Klassensignatur der Klasse Counter. Hinzu kommt, dass die Simulation die Gäste nur zwischen Buffets und Queues hin und herschickt und wir plötzlich in einem Dilemma stecken: Wir haben nicht mehr nur eine einzige Kassenqueue, sondern es kann mehrere geben und der Gast muss sich entscheiden, wo er ansteht. Das wäre noch realisierbar, doch wer handhabt das Öffnen und Schliessen der Kassen samt Umverteilung (Der Restaurantleiter gibt nur die Order, verteilt aber nicht die Kunden um). Wir müssen also eine Schicht dazwischenschalten, welche die Gäste an die richtigen Kassen leitet + sie umverteilt und welche Personal an Essensausgaben abstellen und wegbeordern kann. Dies können wir mit dem Open/Close Prinzip nicht mehr erreichen, da die neuen Klassen keinen gemeinsamen Nenner (Früher Counter) mehr haben. Es sieht so aus, als bräuchte man neue Klassen die Counters und Queues verwalten und durch den Restaurantleiter angesprochen werden können.

## Welche Vorkehrungen hätte es zur Vorkehrung gebraucht

Hätte man Bereits im Vorfeld weiter abstrahiert und Kassen, Essensausgaben und Buffets unterschieden, hätte man das Problem wahrscheinlich verhindern können (Ob man die Queue auslagert oder in der jeweiligen Klasse belässt ist schlussendlich eine reine Designentscheidung:

* Buffet: Hat eine interne Queue und arbeitet einfach die Kunden ab
* Komposition: Beinhaltet Queues und Counters und verwendet ein Standardverhalten oder aber überschreibt dieses
* Essenskomposition: Verwendet eine Queue mit einem Counter und ermöglicht es, Personal abzustellen und auch wieder wegzubeordern
* Kassenkomposition: Verwendet mehrere Counters mit je einer Queue. In der Komposition können weitere Kassen geöffnet und auch welche wieder geschlossen werden – die Komposition übernimmt dann die Umverteilung der Gäste

Alle Kompositionen bieten eine queue Methode an, welche durch Erweiterung des Standardverhalten erweitert werden kann.

## Diskussion über die Vor- und Nachteile

Die Komposition löst das Problem sehr sauber und ermöglicht auch zukünftige Open/Close konforme Änderungen (zum Beispiel, wenn ein Mitarbeiter plötzlich am Buffet aushelfen muss, kann dies in der Komposition, respektive der Erweiterung der Komposition, berücksichtigt werden), doch fügt sie mehr Komplexität hinzu und zerstört den ursprünglich via DDD realisierten Entwurf.

Sie vergrössert auch den LRG, da man normalerweise nur an Buffets, Essensausgaben und Kassen denkt – nicht aber an eine «dazwischengeschaltete» Schicht, welche einem das Denken abnimmt, wo man sich anstellt. Die eigene Vorstellung der Objekte divergiert also mit der Umsetzung und ein Gap entsteht.

## Weitere Überlegungen

Generell könnte man sagen: Abstraktion löst jedes Problem, führt aber zu mehr Komplexität und einem schlechteren LRG.