WBS TRAINING

GoF Strategiemuster

OP2 - Komplexere Anwendungen mit UML entwerfen und entwickeln

WBS TRAINING AG
Lorenzweg 5
D-12099 Berlin
Amtsgericht Berlin HRB 68531
Sitz der Gesellschaft: Berlin

Vorstand:
Heinrich Kronbichler,
Joachim Giese
Aufsichtsrat (Vorsitz): Dr. Daniel Stadler
USt-IdNr.: DE 209 768 248

GLS Gemeinschaftsbank eG
IBAN: DE18 4306 0967 1146 1814 00
BIC: GENODEM1GLS

DQS-zertifiziert nach
DIN EN ISO 9001:2015 Reg.-Nr. 015344 QM15
Zulassung nach AZAV Reg.-Nr. 015344 AZAV

WBS TRAINING

Inhalt

1 Strategiemuster / Strategy pattern 2

1.1 Kategorie: 2

1.2 Aufbau Strategiemuster 2

1.3 Einsatzzweck Strategiemuster 2

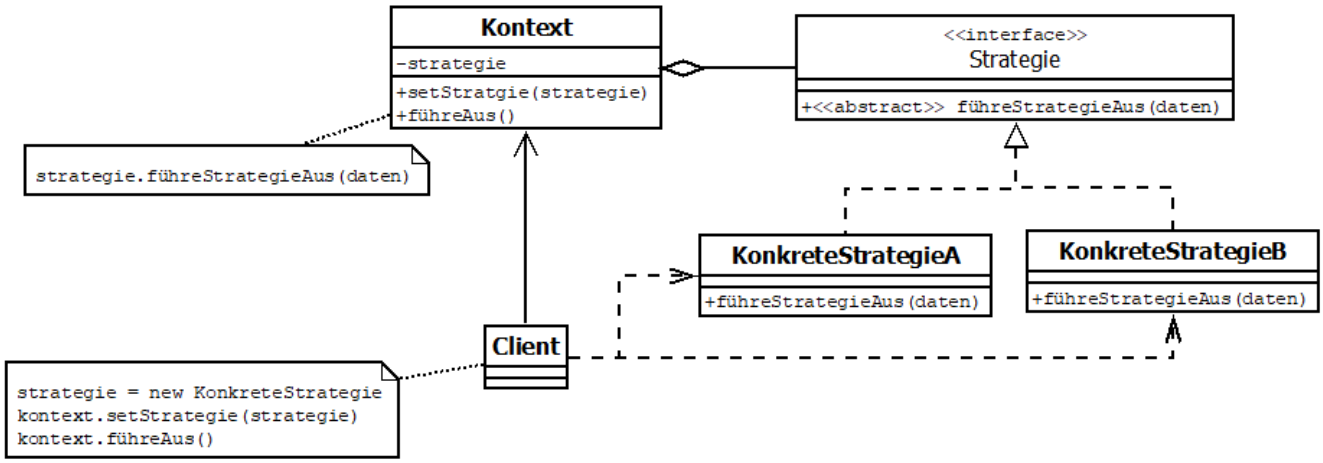
1.4 Implementierungsschritte 3

1 Strategiemuster / Strategy pattern

1.1 Kategorie:

Verhaltensmuster

1.2 Aufbau Strategiemuster



1.3 Einsatzzweck Strategiemuster

- Das Strategiemuster ist ein Verhaltensmuster, welches uns erlaubt eine Familie von Algorithmen zu definieren und jeden Algorithmus in eine separate Klasse kapselt und dadurch die Objekte austauschbar macht.

- Wir verwenden das Strategiemuster, wenn wir verschiedene Varianten (Strategien) eines Algorithmus brauchen und diese zur Laufzeit austauschen möchten.

- Im Strategiemuster kann das Verhalten einer Klasse oder deren Algorithmen zur Laufzeit ausgetauscht werden. Wir kreieren Objekte, die verschiedene Strategien repräsentieren und ein Kontextobjekt dessen Verhalten sich nach je nach eingesetzter konkreter Strategie ändert.

- Wir isolieren die Implementationsdetails der Strategien von dem Code, der die Strategien ausführt.

- Mithilfe des Strategiemuster setzen wir das Open/Closed Prinzip um. Es können einfach neue Strategien hinzugefügt werden (Open: offen für Erweiterungen), ohne bestehenden Code aufwendig ändern zu müssen (Closed: geschlossen für Änderungen).

- Hat das Programm nur wenige Algorithmen, die sich selten ändern, gibt es keinen ausreichenden Grund das Strategiemuster zu implementieren und dadurch das Programm mit neuen Klassen und Interfaces zu verkomplizieren.

- Nachteil: Der Client muss um die Unterschiede der konkreten Algorithmen wissen, um den angemessenen auszuwählen.

- Einige moderne Computersprachen unterstützen funktionale Programmierung und können dieselbe Funktionalität, ohne zusätzliche Klassen und Interfaces, einfacher umsetzen. In Java können wir das erreichen mit Hilfe von Lambdaausdrücken.

1.4 Implementierungsschritte

1. Erstelle ein allgemeines Strategie Interface mit einer abstrakten Methode.

2. Erstelle konkrete Klassen, die das Interface implementieren. Für jede konkrete Strategie eine Klasse.

3. Erstelle eine Kontextklasse. Die Kontextklasse muss ein Attribut bereithalten, um eine Referenz zu einer Strategie abspeichern zu können.

4. Die Kontextklasse funktioniert mit allen Strategien durch dasselbe Interface, welches nur eine Methode bereitstellt, um die gewählte Strategie auszuführen. Durch das Strategiemuster wird die Kontextklasse unabhängig von konkreten Strategien.

5. Das Strategie-Muster unterstützt das Open/Closed Prinzip d.h. der Entwickler kann beliebig neue Algorithmen / Strategien hinzufügen, ändern oder erweitern, ohne die Kontextklasse oder bestehende Strategieklassen anpassen zu müssen.