STATE PATTERN

Konzept

Grundsätzlich gilt, dass das Verhalten eines Objekts abhängig von seinem Zustand ist. Durch die übliche Implementierung soll vermieden werden, dass die Zustände eines Objekts und das davon abhängige Verhalten in einer großen switch-Anweisung (basierend auf enumerierten Konstanten) zu kodieren. Jeder Fall der switch-Anweisung soll in einer eigenen Klasse implementiert werden, so dass der Zustand des Objektes selbst wieder ein Objekt ist, das unabhängig von anderen Objekten ist.

Das Problem von hoher Komplexität und Fragmentierung der Logik wird durch das State Pattern gelöst. Die Ablauflogik für den nächsten Zustand wird isoliert.

Vorteile

Neue Zustände können einfach ins System integriert werden, ohne bestehenden Code zu ändern. Durch die Auslagerung der Zustände in eigene Klassen, bleibt der Code übersichtlich und verständlich. Dadurch können die Zustände auch von anderen Objekten verwendet werden.

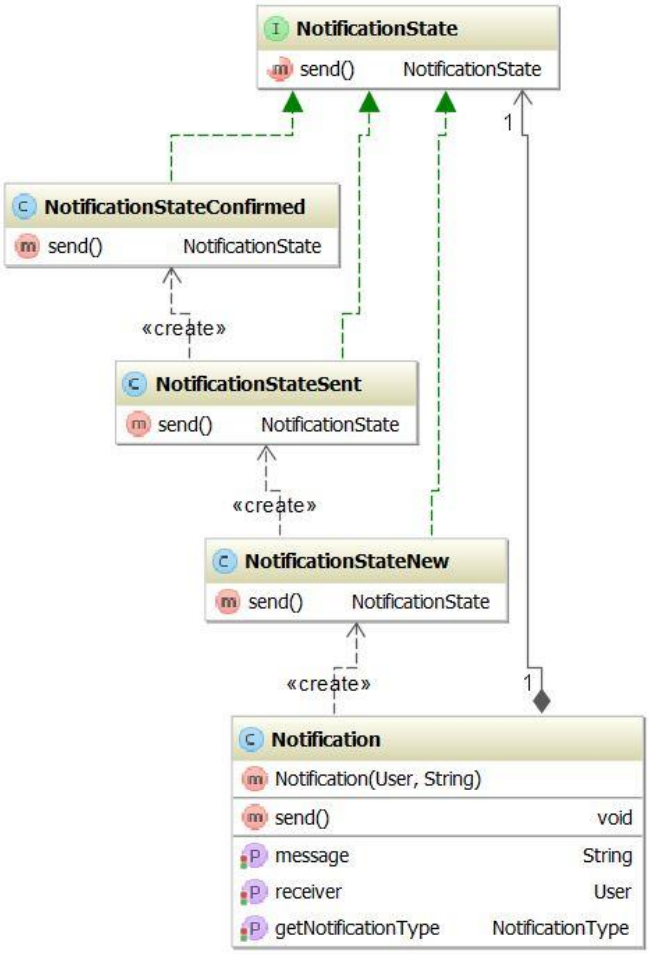
Nachteile

Es werden mehr Klassen benötigt, dadurch entsteht bei der Implementierung ein viel höherer Aufwand. Da keine Select/Case oder If/Else Konstrukte vorliegen, ist es weniger fehleranfällig.

Anwendungsfälle

Das State Pattern wird verwendet, wenn klar spezifiziert ist, welcher Zustand welche anderen Zustände annehmen kann. Also wenn ein zustandsabhängiges Verhalten vorliegt.

In unserem Fall implementieren wir das State Pattern beim Workflow der Benachrichtigung.


NotificationState
send() NotificationState
1
NotificationStateConfirmed
send() NotificationState
«create»
NotificationStateSent
send() NotificationState
«create»
NotificationStateNew
send() NotificationState
«create»
1
Notification
Notification(User, String)
send() void
message String
receiver User
getNotificationType NotificationType