Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Hausarbeit - Software-Design-Pattern

Anpassung - Adapter, Facade, Bridge

Jan Rausch

08. Oktober 2009

Seminarleiter

Dipl. Inf. Frank Schumacher
Problemseminar - Software Design Patterns - Sommersemester 2009
Universität Leipzig

Inhaltsverzeichnis

1 Einleitung **2**

2 Adapter **2**
2.1 Problemstellung . . . 2
2.2 Lösung . . . 2
2.2.1 Objektadapter . . . 3
2.2.2 Klasssenadapter . . . 3
2.3 Vorteile und Nachteile . . . 4
2.3.1 Objektadapter . . . 4
2.3.2 Klassenadapter . . . 4
2.4 Beispiel . . . 4

3 Bridge **5**
3.1 Problemstellung . . . 5
3.2 Lösung . . . 5
3.3 Vorteile und Nachteile . . . 6
3.4 Beispiel . . . 6

4 Facade **7**
4.1 Problemstellung . . . 7
4.2 Lösung . . . 7
4.3 Vorteile und Nachteile . . . 8
4.4 Beispiel . . . 8

5 Abgrenzung der Klassen zueinander **9**
5.1 Entscheidungszeitpunkt für das Muster . . . 9
5.2 Beeinflussung durch das Muster . . . 9
5.3 Einfluss auf die Performance . . . 9

6 Anhang **10**

1 Einleitung

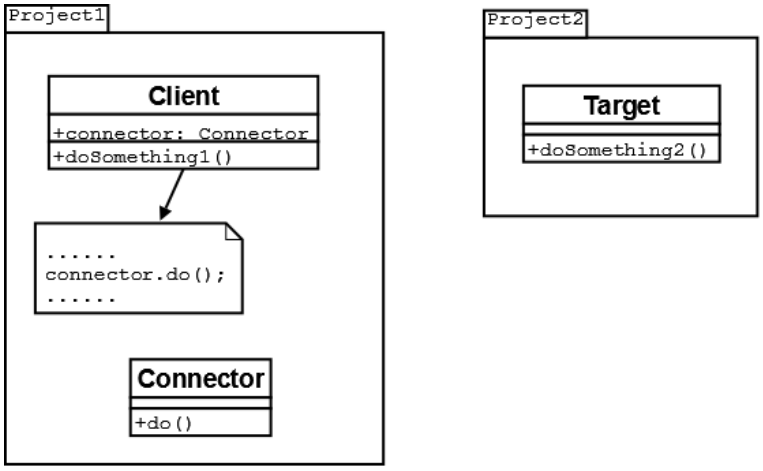
Bei den Design Patterns Adapter, Facade und Bridge handelt es sich um Entwurfsmuster der Kategorie Strukturmuster. Diese Art der Muster liefert Lösungsansätze für die Vereinfachung der Beziehungen der verschiedenen Entitäten des Softwaredesigns, ohne dabei auf spezielle Probleme fixiert zu sein. Entwurfsmuster können also in einem breiten Spektrum an Softwareprojekten auftauchen bzw. verwendet werden.

2 Adapter

In Langenscheidts Fremdwörterbuch [3] findet man unter Adapter folgenden Eintrag: "Ad·ap·ter, der; -s,- TECHNIK Zwischenstück zum Verbinden von an sich unvereinbaren Geräteteilen". In der Softwaretechnik kennt man zwar keine Geräte, aber inkompatible Schnittstellen unterschiedlicher Projekte und genau hier soll das Design-Patttern "Adapter" ansetzen.

2.1 Problemstellung

Ein Ziel der Softwaretechnik ist die Wiederverwendbarkeit von Klassen. Dabei entstehen zwangsläufig Probleme bei der Interaktion von Klassen unterschiedlicher Projekte. Wie auch im technischen Bereich besitzen Projekte meist wohldefinierte Schnittstellen nach außen. Diese sind meistens auf bestimmte Probleme bzw. deren Lösungen optimiert. Es ist im Allgemeinen nicht möglich alle Eventualitäten abzudecken. Vor allem wenn 2 unabhängige Projekte, die aber selber nicht verändert werden können, miteinander kommunizieren/arbeiten sollen, kann eine Inkompatibilität erwartet werden.

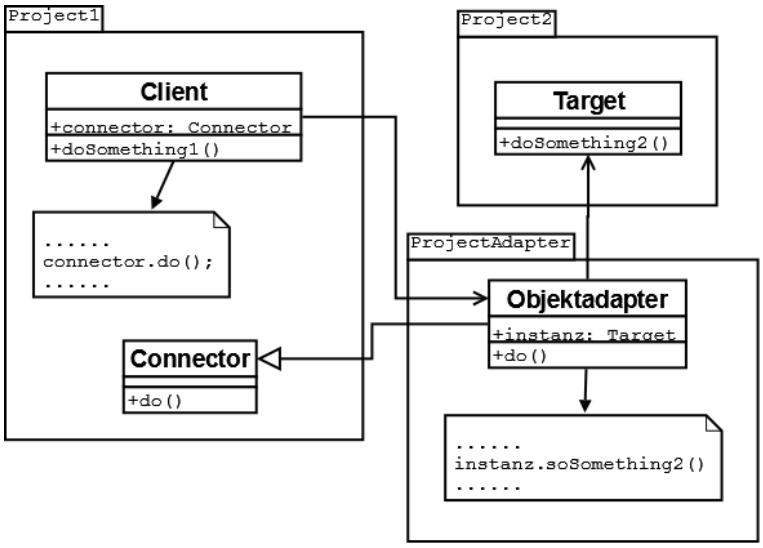


2.2 Lösung

Die Autoren von [2] schlagen 2 verschiedene Lösungen vor. Erstens, eine zusätzliche Klasse einzuführen, die mittels Vererbung in den Clienten einklinkt und dann entsprechend der benötigten Aufgabe Befehle an eine Instanz des Ziels delegiert (Objektadapter). Zweitens, eine zusätzliche Klasse einzuführen, die mittels Mehrfachvererbung zum einen die Methoden des Ziels erbt als auch die benötigte Schnittstelle der Ausgangsklasse (Klassenadapter).

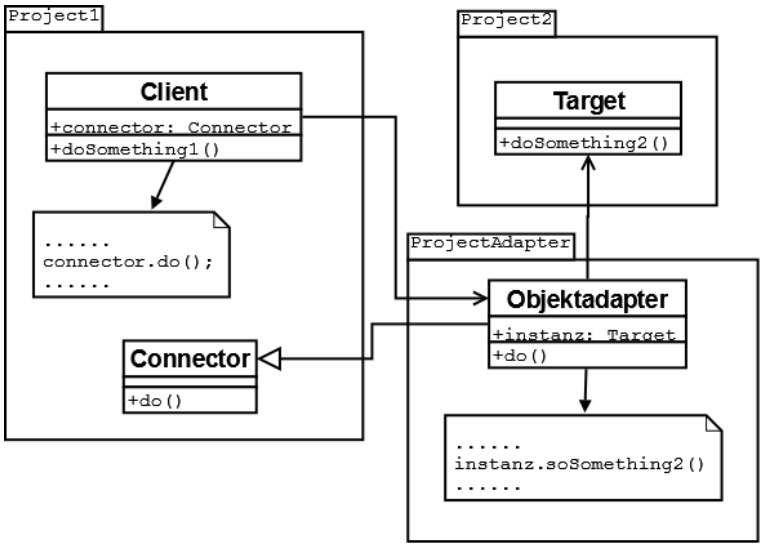
2.2.1 Objektadapter

Wie bereits einleitend erwähnt, benutzt dieser Adaptertyp einfache Vererbung und Delegation an eine Objektinstanz. Ziel der Vererbung ist es, dass sich der Adapter als eine für den "Client" kompatible Klasse ausgibt. Ziel der Delegation ist, dass die Methodenaufrufe des Clients auf das "Target" umgesetzt werden.



2.2.2 Klasssenadapter

Der Klassenadapter nutzt Mehrfachvererbung. Dabei erbt er ebenfalls von dem "Connector", aber zusätzlich erbt er auch von dem "Target". Statt also ein Objekt zu referenzieren, ist der Klassenadapter sowohl "Connector" als auch "Target". Der Klassenadapter delegiert also die Methodenaufrufe intern.



2.3 Vorteile und Nachteile

Für beide Adapter gibt es gemeinsame Vor- und Nachteile. Der Hauptvorteil ist natürlich durch das Problem bzw. die Lösung desselben gegeben. Ein Adapter erhöht die Wiederverwendbarkeit von Softwareprojekten. Daraus ergeben sich auch die üblichen Folgen: geringere Kosten, kürzere Entwicklungszeiten und länger getestete Software. Nachteilig wirkt sich vor allem die zusätzliche Delegationsschicht aus, da diese zusätzliche Operationen ausführen muss, die bei einer vollständig selbstständigen Lösung nicht anfallen würden. Dieser Nachteil resultiert aber eher aus der Problemstellung, welche ja gerade sagt, dass die beiden beteiligten Projekte nicht geändert werden dürfen.

2.3.1 Objektadapter

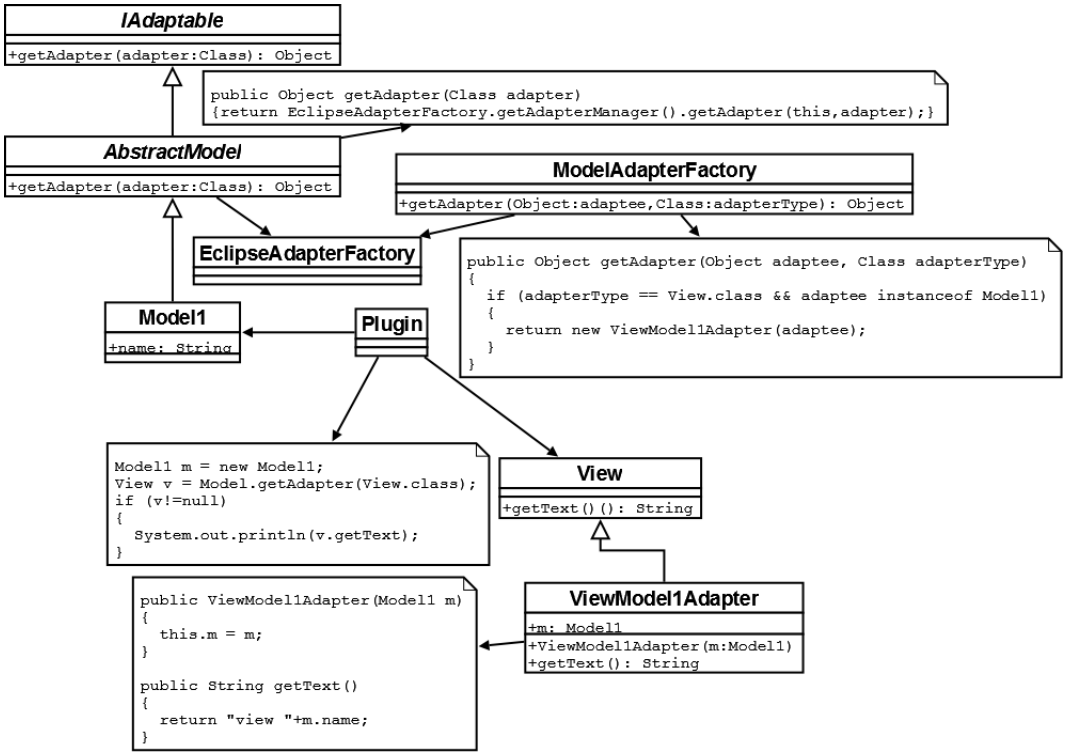
Der Objektadapter verzichtet auf Mehrfachvererbung und kann deswegen in nahezu allen objektorientierten Programmiersprachen umgesetzt werden.

2.3.2 Klassenadapter

Da der Klassenadapter auf Mehrfachvererbung setzt, kann er zum Beispiel nicht in Java umgesetzt werden. Da dieser Adapter aber von allen Klassen erbt, kann er auch auf die internen Methoden aller Klassen zurückgreifen. Dies ist von Vorteil, wenn die Ausgangsklassen bereits einen Teil der benötigten Logik implementiert haben.

2.4 Beispiel

Wie in [4] beschrieben, setzt Eclipse das Adapter Pattern in großem Umfang ein. Ziel ist es hier Model und View zu trennen. Der Adapter ist der Mittler zwischen Model und View und meldet sich mittels XML Konfiguration bei Eclipse an und wird von einer Factory zur Verfügung gestellt.

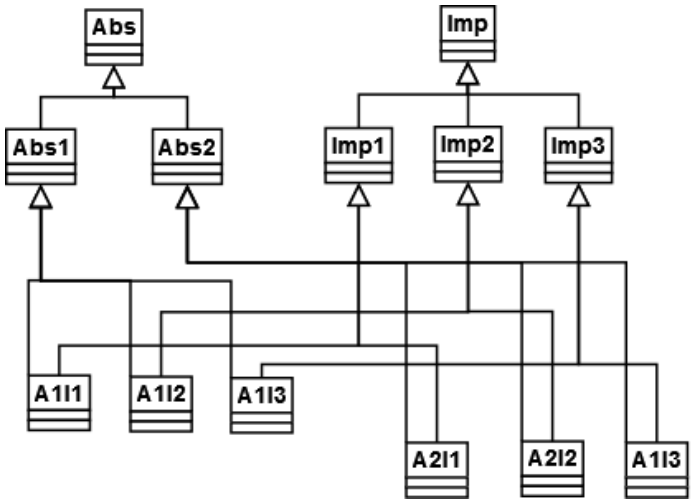


3 Bridge

Eine Bridge bzw. Brücke bezeichnet ein künstlich erschaffenes Objekt, welches 2 zumeist eigenständige Objekte verbindet.

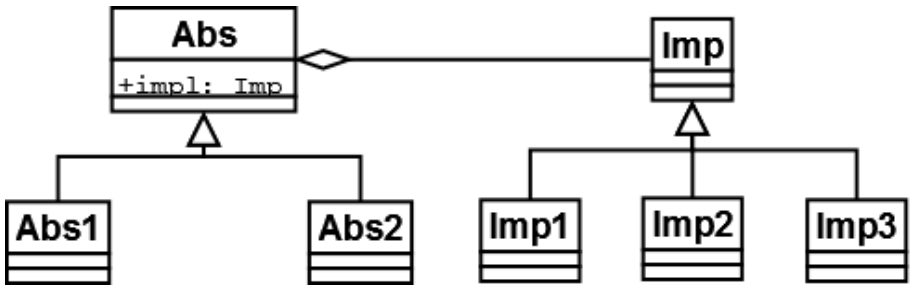
3.1 Problemstellung

Im Normalfall ist die Vererbungsstruktur ein eng gekoppelter Klassengraph. Für bestimmte Probleme kann es aber sinnvoll sein, mehrere parallele Vererbungshierachien zu haben. In einer Programmiersprache mit Mehrfachvererbung würde das zu einer großen und unüberschaubaren Menge (bis zu Anzahl "Produkt aller Summen der Klassen in den Hierachieen") an zusätzlichen Klassen führen, welche sich nur durch andere Überklassen unterscheiden. In einer Programmiersprache ohne Mehrfachvererbung könnte man nur eine Hierachie direkt umsetzen und müsste alle weiteren benötigten Methoden für jede einzelne Zielklasse erneut implementieren.



3.2 Lösung

Im einfachsten Fall besteht das Problem aus nur 2 Hierachien. Dieses entspricht dann auch dem in [2] vorgestellten Problem. Die Autoren sprechen hier von Abstraktion und einer davon unabhängigen Implementierung. Erreicht wird dies, indem mittels Aggregation auf die genaue Implementierung zugegriffen wird.



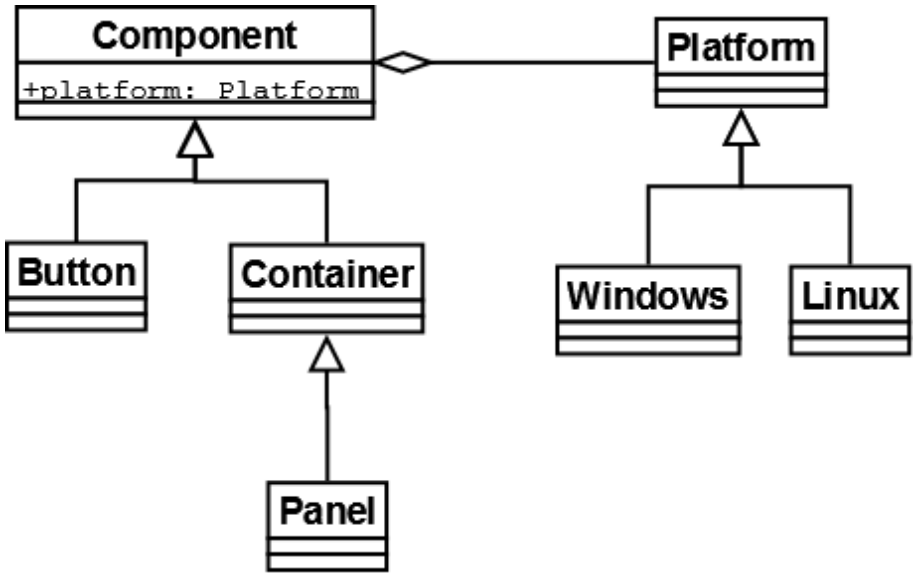
3.3 Vorteile und Nachteile

Die Bridge erhöht die Übersichtlichkeit der Modelierung und ermöglicht es auch Mehrfachvererbungen in Programmiersprachen ohne Mehrfachvererbung zu simulieren. Ein Nebeneffekt der Bridge ist, dass Abstraktion und Implementierung enkoppelt werden. Die Implementierung kann also beliebig, auch zur Laufzeit, ausgetauscht werden und damit den tatsächlichen Begebenheiten angepasst werden.

Als Nachteil könnte man höchstens werten, dass es nun nicht mehr möglich ist, Abstraktion und Implementierung exakt aufeinander abzustimmen und dadurch eventuell Perfomanzeinbußen entstehen. Dies ist aber wohl eher ein konstruiertes Problem, als tatsächlich existent.

3.4 Beispiel

Ein Beispiel für den Einsatz von Bridge Pattern, ist die Java AWT. Da es eine Java-Laufzeitumgebung für viele verschiedene Betriebssysteme (und damit auch Windowsmanager) gibt, muss es auch verschiedene Implementierungen für die GUI Elemente geben. In der AWT besitzt jede Komponente ein Attribut für die plattformabhängige Implementierung und nutzt diese für die Darstellung zur Laufzeit.

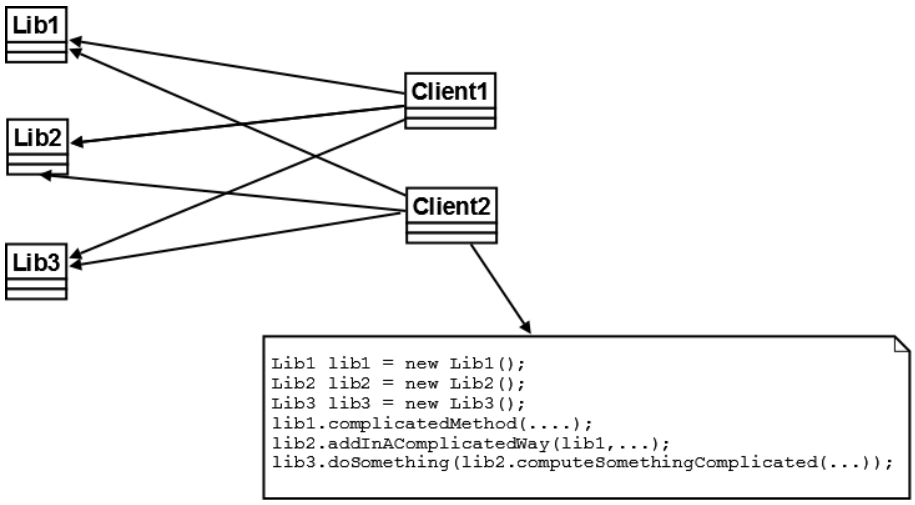


4 Facade

Laut digitalem Wörterbuch der Deutschen Sprache [1] handelt es sich bei einer Facade bzw. Fassade um entweder die "Front, Ansicht eines Gebäudes" oder "abwertend: die sichtbare, äußere Seite". Natürlich existieren in der Informatik weder reale Gebäude noch benötigt man abwertende Begriffe zur Problemlösung. Es existieren aber unüberschaubare Bibliotheken und Projekte bei denen ein Ausschnitt des Ganzen, den Umgang mit diesen vereinfachen würde.

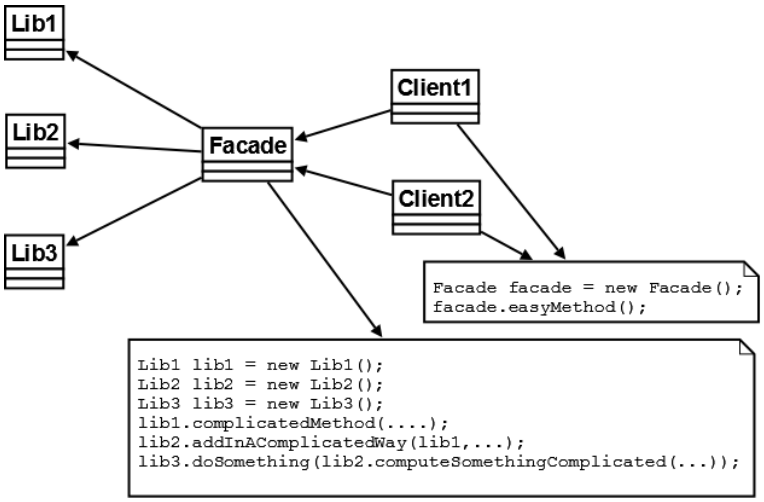
4.1 Problemstellung

Wenn Software bzw. Bibliotheken "reifen", verbessert sich meist nicht nur die Güte sondern auch der Umfang. Anfänglich überschaubare Projekte wirken auch bei sorgfältiger Strukturierung und Dokumentation für einen neuen Betrachter irgendwann wie eine Lawine, die nur vor hat ihn zu erdrücken. Von der Fülle an Möglichkeiten und Schnittstellen kristallisieren sich aber meist bestimmte, häufig benötigte Aufgaben bzw. Abläufe heraus. Das Problem wäre nun also: Wie kann man das Benutzen von komplexen Projekten vereinfachen?



4.2 Lösung

In [2] wird vorgeschlagen mittels einer zusätzlichen Klasse, eine Menge an einheitlichen und vereinfachten Schnittstellen zur Verfügung zu stellen. Ziel dieser Facade-Klasse ist es also, Anfragen entgegenzunehmen und zu delegieren. Die Facade hat aber nicht die Aufgabe, dass dahinterliegende System zu verstecken, es soll also auch weiterhin benutzt werden können.



4.3 Vorteile und Nachteile

Die Facade vereinfacht die Verwendung von komplexen Systemen und ermöglicht es auch unerfahrenen Entwicklern auf komplizierte aber stabile Lösungen zurückzugreifen. Je mehr die Facade statt der eigentlichen Projekte genutzt wird, umso unabhängiger wird das zu entwickelnde Projekt auch von diesem.

Der Hauptnachteil ist die weitere Indirektionsstufe, die eingeführt werden muss.

4.4 Beispiel

Die Facade ist ein Pattern, welches oft implementiert wird, ohne das Bewusstsein dass es sich um eine Facade handelt und auch als Benutzer einer Facade ist man sich meistens nicht darüber bewusst, dass es sich um eine solche handelt. Wichtige allgemeine Einsatzbereiche sind Dateioperationen, Grafikprogrammierung und mathematische Berechnungen. Statt also eines der zahlreichen Beispiele aus der Welt der Informatik zu nehmen, soll auf den Anhang (Abbildung 1) verwiesen werden, indem nicht ganz ernstgemeint auf die Vorteile einer Facade eingegangen werden soll.

5 Abgrenzung der Klassen zueinander

Wie bereits Eingangs erwähnt, handelt es sich bei diesen 3 Mustern um Strukturmuster. Dabei ist es aber wichtig, neben den Gemeinsamkeiten vor allem die Unterschiede klar hervorzuheben. Dies soll im folgenden Abschnitt geschehen.

5.1 Entscheidungszeitpunkt für das Muster

Der Adapter ist das Muster, welches zu einem beliebigen Zeitpunkt nach Fertigstellung der zu verknüpfenden Ausgangsprojekte eingesetzt werden kann. Einen Adapter zu verwenden, wenn man Einfluss auf eines der beiden beteiligten Projekte nehmen kann, ist aber nicht sinnvoll, da es bessere Möglichkeiten gibt, welche auch ohne die zusätzliche Delegationsschicht auskommen.

Die Entscheidung für die Bridge muss bereits zum Beginn des Softwareentwurfs getroffen werden.

Die Facade kann zu einem beliebigen Zeitpunkt erstellt werden. Für den Anwender der Facade ist es natürlich notwendig, dass die Facade vor Beginn des eignenen Projektes entworfen wird.

5.2 Beeinflussung durch das Muster

Der Adapter beeinflusst keines der beteiligten Teilprojekte. Beide können und sollen unabhängig von dem Adapter existieren. Die Bridge hingegen ist elementarer Bestandteil des Projektes und auch wenn die Klassen unabhängig von den speziellen anderen Klassen sind, so ist das Gerüst der Bridge trotzdem fest mit dem Projekt verbunden. Die Facade hat keinen Einfluss auf die Ausgangsklassen. Das Projekt welches die Facade aber verwendet, steht in starker Abhängigkeit zu dieser.

5.3 Einfluss auf die Performance

Im Allgemeinen kann man erwarten, dass der Adapter den negativsten Einfluss auf die Performance des Gesamtsystems hat. Wenn die Möglichkeit exisiert 2 Systeme direkt zu verknüpfen, sollte man immer diesen Weg gehen, da man nur so Einfluss auf eventuell unnötige, redundante Arbeit in den Ausgangsprojekten nehmen kann. Der Einfluss der Facade auf die Performance ist stark problemabhängig. Wenn die Facade genau die selben Aktionen ausführt, die man auch im Optimalfall ausführen würde, hat diese nahezu keinen Einfluss auf die Performance. In allen anderen Fällen aber schon. Man muss sich an dieser Stelle nur einmal eine Methode zum Zeichnen von Objekten vorstellen, bei der die Facade im Optimalfall Hardwarebeschleunigung etc. selber aktiviert und in einem anderen Fall die Facade starr auf Hardwarebeschleunigung verzichtet. Bei der Bridge ist am wenigsten mit einem negativen Einfluss auf die Performance zu rechnen. In bestimmten Fällen, kann das Austauschen zur Laufzeit (und der daraus eventuell resultierenden optimaleren Implementierung), zu einer Verbesserung der Performance verglichen mit dem eigentlichen Design führen.

6 Anhang

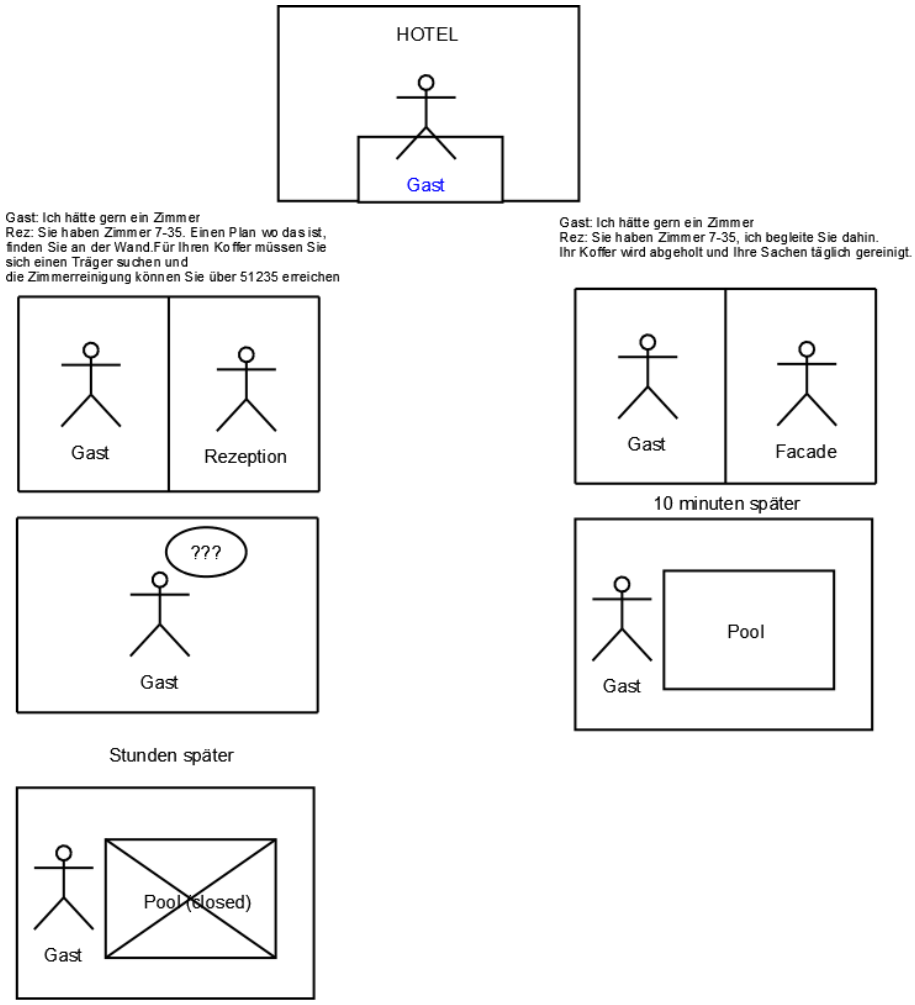


Abbildung 1:

Literatur

[1] Berlin-Brandenburgische Akademie der Wissenschaften. Das digitale wörterbuch der deutschen sprache des 20.jh. 2003.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addisson-Wesley, Toronto, Ontario. Canada, 1995.

[3] Langenscheidt KG. Langenscheidt fremdwörterbuch. Berlin und München, Germany, 2009.

[4] Jeffrey Ricker. Eclipse adapters - a hands-on, hand-holding explanation. 2007.