4. Softwaretechnik und Softwareentwicklung

Die Softwaretechnik beschäftigt sich mit der Umsetzung und Strukturierung von Projekten, in denen man eine Software für eine konkrete Problemstellung programmiert. Die Prinzipien der Softwaretechnik lassen sich jedoch auf alle denkbaren Projekte und Teamaufgaben übertragen.

4.1 Design Pattern

In der Informatik gibt es gewissen Problemstellungen, die immer wieder vorkommen. Manche Aufgaben müssen in so vielen Programmen irgendwo erfüllt werden, dass sich über die Jahre hinweg bewährte Lösungen als Standardlösung etabliert haben. Diese Standardlösungen nennen wir ***Design Pattern***.

Vorteile von Design Pattern:

- Häufig getestet → keine „Kinderkrankheiten“ mehr
- Nicht offensichtliche Probleme der Aufgabe sind gelöst
- Lösung leicht verständlich für jemand, der nicht bei der Entwicklung dabei war (oft sind Support und Entwickler in einer Firma verschiedene Abteilungen)
- Kein unnötiger Aufwand um ein bereits bekanntes Problem zu lösen

Wir kennen bereits zwei Design Pattern, das Kompositum (für die Liste/Baum/Graph Stichwort ABSCHLUSS) und das Adapter Pattern (Oberklasse, die z.B. interne Liste versteckt).

Im folgenden Kapitel lernen wir noch zwei weitere Pattern, die, im Gegensatz zu den doch sehr speziellen Pattern Kompositum und Adapter, in der Softwaretechnik bei der Entwicklung **jedes** Programms angewendet werden[1].

4.1.1 Beobachter

Das Beobachter Pattern begegnet uns jeden Tag, wenn wir surfen. Betrachten wir z.B. Youtube und sein Abonnementsystem: Man kann mit einem Knopfdruck immer über neue Videos auf einem Kanal der Wahl auf dem Laufenden bleiben.

Der User muss also irgendwie Nachrichten über neue Inhalte bekommen und der „Subscriptionservice“ muss allen, die abonniert haben eine Nachricht zukommen lassen. Überlegen wir uns kurz, welche Bestandteile als der USER und der SUBSCRIPTIONSERVICE haben muss:

---

1 Der Superlativ an dieser Stelle ist so zu verstehen: De facto ist es so, dass beide Pattern in jeder guten Software enthalten sein sollten. Oft lassen Programmierer bei einfachen Aufgaben die Pattern trotzdem weg. Das ändert nichts daran, dass die Software damit schlechter zu warten ist, als wenn die Pattern verwendet worden wären.

USER:

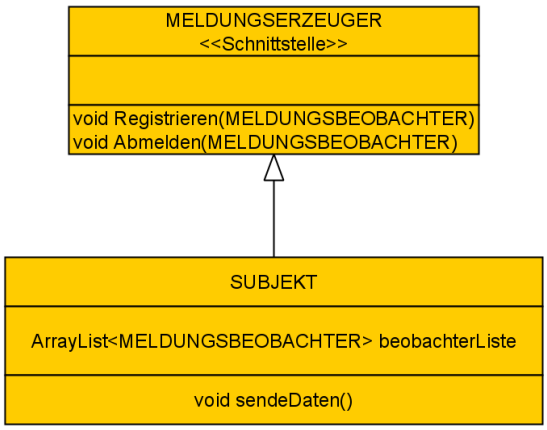
- Muss Daten empfangen können (z.B. Notifications)

SUBSCRIPTIONSERVICE:

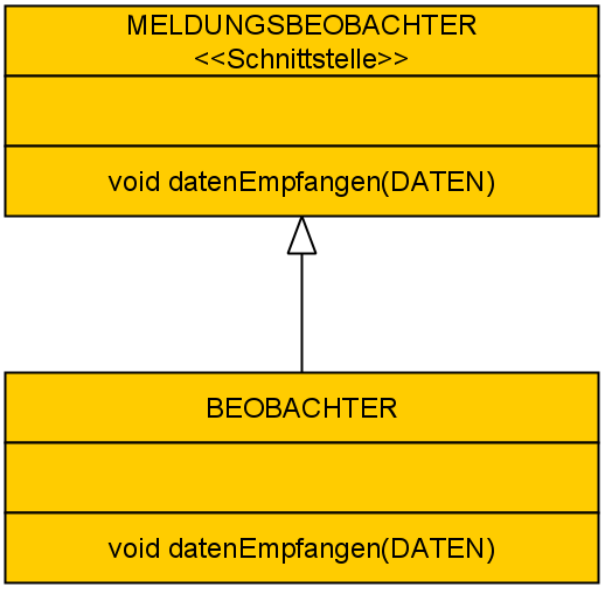
- Muss Daten senden können
- Muss wissen, wer die die Daten empfangen will
- Man kann abonnieren
- Man kann sein Abo abbrechen

Die Basis Variante des Beobachter Pattern beinhaltet nur Schnittstellen, die das Anmelden/Abmelden und das Empfangen der Daten enthält. Wir schauen uns direkt eine Variante an, die diese Schnittstellen in Klassen konkretisiert.

Das führt uns zu folgenden Klassendiagrammen:



Das Interface MELDUNGSERZEUGER muss in erster Linie nur Schnittstellen mit denen sich MELDUNGSBEOBACHTER an- und abmelden können bereitstellen.
Die Konkretisierung des MELDUNGSERZEUGERS (oft SUBJEKT genannt) hält eine erweiterbare Liste (Java: ArrayList) von Beobachtern, die über neue Daten informiert werden wollen. Es hat eine Methode, die allen Beobachtern die neuen Informationen zuschiebt.



Die Schnittstelle MELDUNGSBEOBACHTER bietet Methoden, die es ermöglichen Nachrichten zu erhalten.

Der Beobachter muss die Methoden der Schnittstelle konkretisieren, indem die einkommenden Daten verarbeitet werden.

Anmerkungen:

Es gibt zwei grundlegend unterschiedliche Designmöglichkeiten für ein SUBJEKT:

1. das Subjekt schickt die Daten direkt an seine Beobachter
2. das Subjekt informiert darüber, dass sich etwas geändert hat und jeder Beobachter holt sich die Informationen selber.

Im ersten Fall ist die Implementierung schwer erweiterbar wenn man nicht eine DATEN Klasse definiert, die erweiterbar ist, falls sich beispielsweise die Daten ändern (z. B. Datentyp, Menge, …).
Manchmal braucht eine Applikation auch nicht alle Daten der DATEN Klasse, was eigentlich bedeutet, dass manche Daten sinnloserweise mitgeschickt werden (ggf. will man diese sogar nicht jedem BEOBACHTER zukommen lassen).
Wenn eine Implementierung mit `ZustandGeandert()` gewählt wurde, muss im BEOBACHTER eine Reaktion auf die Benachrichtigung, dass es neue Daten gibt festgelegt werden.
Daher ist es oft gut den zweiten Weg zu gehen und eine boolsche Methode ZustandGeaendert() einzubauen und mehrere Methoden in der Schnittstelle bereitzustellen, über die die Beobachter eine Kopie der Daten bekommen können. Je nach Anwendungsfall muss man entscheiden, welche Version man umsetzt.
In unserem Beispiel wählen wir die Variante 1, da diese für alle Standardfälle (und vor allem für die Schule) ausreichend ist.

Umsetzung in Java

Alle Methoden der Klassen erfordern keinen komplizierten Algorithmus. Allerdings müssen wir kurz über die Struktur und Methoden der ArrayList reden.

Die ArrayList wird als Attribut wie folgt angegeben:

```
ArrayList<MELDUNGSBEOBACHTER> beobachterListe;
```

Der Klassenbezeichner in den „Brackets“ („<“ und „>“) gibt dabei an, welchen Datentyp die Daten der Liste haben.

Typische Methoden wie das Hinzufügen („`add(Element)`“), Entfernen („`remove(Element)`“ und `remove(int anStelle)`), Löschen der Liste („`clear()`“), Suchen („`indexOf(Element)`“), Kopieren („`clone()`“), Prüfen ob ein Objekt in der Liste ist („`contains(Element)`“), Ersetzen an eine Stelle(„`set(int index)`“), Länge der Liste („`size()`“) und das Ausgeben eines Elements der Liste („`get(int index)`“) sind umgesetzt.

Zusätzlich gibt es eine schöne Abkürzung um das Benachrichtigen aller BEOBACHTER in der `ArrayList` schneller zu machen.

Die Wiederholung über die Länge der `ArrayList` lässt sich abkürzen mit:

```
for (MELDUNGSBEOBACHTER b: beobachterListe)
  {
     //hier werden daten Übertragen:
     b.datenEmpfangen(daten);
  }
```

Die Wiederholung geht damit automatisch über alle Elemente in der `ArrayList`. Jedes Element kann innerhalb der Wiederholung mit dem Bezeichner `b` angesprochen werden. Dabei hat man jedoch keinen Zugriff auf den Index (braucht man hier aber auch nicht…).

Aufgaben:
1) Kopieren Sie die Vorlage Beobachter aus dem Klassenordner.

2) Ergänzen Sie die Schnittstellen MELDUNGSERZEUGER und MELDUNGSBEOBACHTER um die vorgegebenen Methoden des Klassendiagramms.

3) Erweitern Sie die Klassen SUBJEKT und BEOBACHTER um die Attribute aus den Klassendiagrammen. Geben Sie zusätzlich dem BEOBACHTER einen `String name` und dem SUBJEKT Daten des Typs `String`.

4) Sorgen Sie dafür, dass SUBJEKT und BEOBACHTER die Schnittstellen MELDUNGSERZEUGER und MELDUNGSBEOBACHTER implementieren. Ergänzen Sie die nötigen (nun geforderten) Methoden und füllen Sie diese.
Geben Sie dabei die Daten weiter und Sorgen Sie dafür, dass SUBJEKT und BEOBACHTER immer in die Console etwas schrieben, wenn sie etwas versenden/empfangen. Nutzen Sie das Attribut name im BEOBACHTER, um die Ausgabe leserlich zu machen (z. B. SUBJEKT sendet an „Hans“ Nachricht „bla“).

5) Überlegen Sie, welche Teile Sie umbauen müssten, um die Implementierung mit „`ZustandGeandert()`“ zu verwenden.