Teachlet zum Entwurfsmuster Zustand: Sessel-o-matic 1.0.0

Autoren

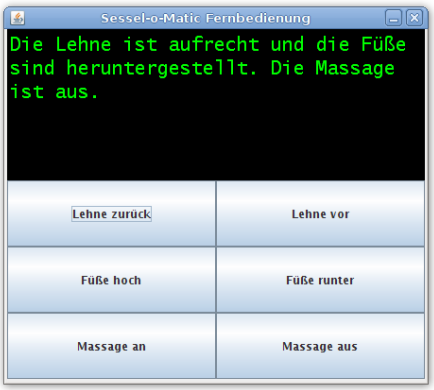
Julian Fietkau, Janina Nemec

Lernziel

Das Ziel ist, das Zustands-Entwurfsmuster anhand einer konkreten Problemstellung kennen zu lernen.

Ausgangssystem

Vorgegeben ist eine Java-(Swing-)Anwendung, die eine Fernsteuerung für einen Massagesessel nachbildet. Über Buttons kann der Zustand des Sessels auf bestimmte Weise verändert werden, wobei zwei der Buttons noch ohne Funktionalität sind.



Aufgaben

Die Aufgabe ist, die mittleren zwei Buttons der Fernbedienung gemäß einer vorgegebenen Spezifikation mit Funktionalität zu versehen. Bisher wird der Zustand des Sessels durch mehrere Zustandsfelder modelliert. Bei der Umsetzung wird klar, dass dieser Ansatz den gesteigerten Anforderungen nicht gewachsen ist und durch eine flexiblere Struktur ersetzt werden muss.

Programmiersprache

Java 1.6

Programmierumgebung

Eclipse 3.4

Zeitumfang

90-120 Minuten

Teilnehmerbewertung

Die durchschnittliche Bewertung des Plenums (16 Stimmen) auf einer Skala von 0 (schlecht) bis 5 (optimal):

- Ausgangssystem: 4,0
- Foliensatz: 4,5
- Eignung für das Lernziel: 4,5

Inhaltsverzeichnis

1. Inhaltsverzeichnis

- **Foliensatz**: Die Präsentationsfolien zum Teachlet liegen im LaTeX- und im PDF-Format vor. Der LaTeX-Code liegt zusammen mit den nötigen Bildern in einer eigenen Zip-Datei. *(Dateiname: ZustandFolien.pdf/.zip)*
- **Ausgangssystem**: Das Ausgangssystem liegt als Eclipse-Projekt gepackt vor, gemeinsam mit dem Zielsystem in einem Archiv, natürlich unabhängig voneinander importierbar. *(Dateiname: ZustandSysteme.zip)*
- **Zielsystem**: Das Zielsystem liegt in der selben Zip-Datei wie das Ausgangssystem. *(Dateiname: ZustandSysteme.zip)*
- **Handout**: Das Handout liegt wie die Folien im LaTeX- sowie im PDF-Format vor. Die LaTeX-Version ist mit den verwendeten Bildern in einem Zip-Archiv gepackt. *(Dateiname: ZustandHandout.pdf/.zip)*
- **Dokumentation**: Auch diese Dokumentation ist im LaTeX- und im PDF-Format vorhanden. Ebenso wie die anderen liegt auch dieses LaTeX-Dokument wieder in einem eigenen Zip-Archiv vor. *(Dateiname: ZustandDokumentation.pdf/.zip)*

In dieser Dokumentation folgt nun eine ausführliche Beschreibung des Teachlets, beginnend mit einer Darstellung des Ausgangssystems und dessen Verhaltens, sowie der zugrundeliegenden Architektur. Im nächsten Abschnitt wird die Aufgabe an die Teilnehmer zusammengefasst. Danach folgt eine ausführliche Choreographie, in der genau beschrieben ist, wie das Teachlet aufgeführt werden kann. Im darauf folgenden Abschnitt wird das mitgelieferte Zielsystem genauer beschrieben sowie kurz auf mögliche alternative Ergebnisse eingegangen. Schließlich folgt eine Auflistung der bisherigen Einsätze sowie eine Zusammenfassung offener Punkte, die vor der nächsten Aufführung bearbeitet werden könnten.

2. Ausgangssystem

Das System, an dem in diesem Teachlet gearbeitet wird, simuliert eine Fernbedienung für einen Massagesessel. Wir beginnen mit der Beschreibung der Oberfläche.

2.1. Oberfläche

Nach dem Start der Software (Klasse `StartUp`) zeigt sich ein Fenster mit einem Textfeld und sechs in einem 3×2-Raster angeordneten Buttons (Abbildung 1).

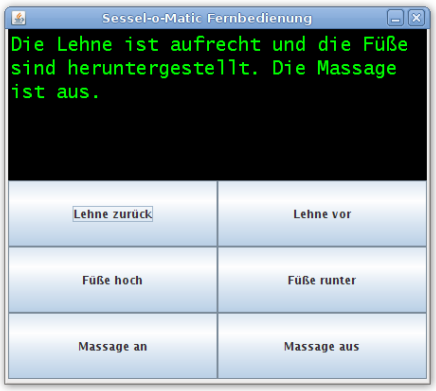


Abbildung 1: Screenshot des Ausgangssystems

Das Textfeld ist eine leidlich realisitische Nachbildung eines kleinen LCD-Bildschirms, wie man ihn auf der fiktiven Sessel-o-Matic-Fernbedienung finden würde. Zwecks guter Lesbarkeit auf einem Beamer wurde eine recht hohe Schriftgröße gewählt. Die sechs Buttons stellen die Knöpfe der Fernbedienung dar, über die die Interaktion mit dem Sessel stattfindet.

2.2. Funktionalität

Die einzige Eingabemöglichkeit sind die sechs Buttons. Der simulierte Sessel hält intern und von außen unsichtbar seinen Zustand. Je nachdem, in welchem Zustand er sich befindet, bewirkt ein Klick auf einen Button, dass sich der Zustand des Sessels und damit der Text auf dem Display ändert oder auch nicht. Hierbei ist die Implementation zunächst unvollständig: Die Buttons „Füße hoch“ und „Füße runter“ zeigen noch keinerlei Wirkung. Durch Betätigen der restlichen vier Buttons lässt sich zügig feststellen, dass der Sessel einen Zustandsraum besitzt, der durch ein recht einfaches Zustandsdiagramm spezifiziert werden kann (Abbildung 2).

Abbildung 2: Zustandsdiagramm des Ausgangssystems

Die Massage kann also nur bei zurückgestellter Lehne aktiviert werden und die Lehne kann nur bei deaktivierter Massage hochgestellt werden.

Der derzeitige Zustand des Sessels wird nach jeder Betätigung eines der Buttons im Textfeld angezeigt.

2.3. Architektur

Das Ausgangssystem besteht aus fünf Klassen (genauer: vier Klassen und einem Interface).

1. `StartUp`: Diese Klasse enthält eine `main`-Methode, über die das System gestartet wird. In dieser Methode werden eine Fernbedienung und ein Sessel erzeugt und miteinander verknüpft.
2. `Fernbedienung`: Diese Klasse erbt von `JFrame` und stellt einerseits die GUI des Systems dar, übernimmt aber auch gleichzeitig die fachliche Funktion der Fernbedienung. Im Konstruktor wird ein Sessel

entgegengenommen, mit dem die Fernbedienung in Zukunft interagieren soll. Ansonsten implementiert `Fernbedienung` das Interface `Display`. Dadurch wird es dem Sessel ermöglicht, Nachrichten über seinen Zustand an die Fernbedienung zu schicken, welche diese dann darstellt.

3. `Display`: Dieses Interface, welches von `Fernbedienung` implementiert wird, dient dazu, einen Abhängigkeitszyklus von Sessel und Fernbedienung zu vermeiden. Am Sessel kann mit der Operation `setzeDisplay` festgelegt werden, wohin der Sessel seine Statusnachrichten sendet. Im Prinzip handelt es sich um eine einfache Implementation des Beobachtermusters mit genau einem Beobachter.
4. `Sessel`: Diese Klasse modelliert den Sessel. Sie verfügt über sechs Methoden, in denen der Sessel auf die verschiedenen Eingaben reagiert. Davon sind im Ausgangssystem zwei noch leer, nämlich genau `stelleFuesseHoch` und `stelleFuesseRunter`. Der Zustand des Sessels wird im Ausgangssystem mittels zweier Exemplarvariablen gehalten, welche private innere Enumerationstypen benutzen (`LehneZustand` und `MassageZustand`). Es exisitert bereits ein `enum` namens `FussZustand`, dieser ist jedoch ungenutzt und erzeugt dadurch in Eclipse eine Warnung.
5. `Nachrichten`: Diese Klasse stellt als öffentliche statische Konstanten die Statustexte bereit. Alle acht Kombinationen der möglichen Statuswerte von Lehne, Fußstütze und Massage sind enthalten. Sie werden vom Sessel und von der Fernbedienung benutzt.

Die Abhängigkeiten zwischen den Klassen lassen sich am besten anhand eines Klassendiagramms veranschaulichen (Abbildung 3).

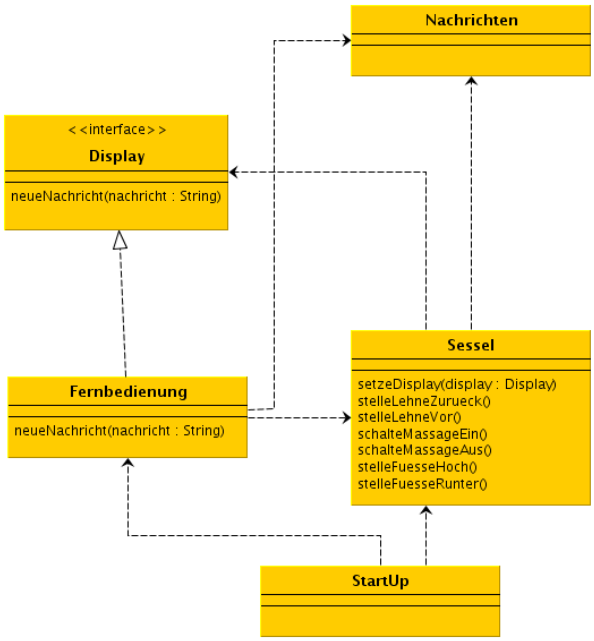


Abbildung 3: Klassendiagramm des Ausgangssystems

Die Klassen `StartUp` und `Nachrichten` haben mit dem Lernziel des Teachlets nichts weiter zu tun und sollten deshalb nicht zu sehr in den Mittelpunkt gestellt werden. Dagegen stellt die Klasse `Sessel` den Dreh- und Angelpunkt des Teachlets dar.

Aufgrund ihrer zentralen Rolle im Teachlet wollen wir die Klasse `Sessel` an dieser Stelle vollständig reproduzieren (Listing 1).

Listing 1: Sessel.java

```
1  public class Sessel
   {
      private enum LehneZustand
      {
5        LEHNE_AUFRECHT, LEHNE_ZURUECK
```

```
    }

    private enum FussZustand
    {
       FUESSE_OBEN, FUESSE_UNTEN
    }

    private enum MassageZustand
    {
       MASSAGE_AN, MASSAGE_AUS
    }

    private Display _display;
    private LehneZustand _zustandLehne;
    private MassageZustand _zustandMassage;

    public Sessel()
    {
       _zustandLehne = LehneZustand.LEHNE_AUFRECHT;
       _zustandMassage = MassageZustand.MASSAGE_AUS;
    }

    public void setzeDisplay(Display display)
    {
       _display = display;
    }

    public void stelleLehneZurueck()
    {
       if (_zustandLehne == LehneZustand.LEHNE_AUFRECHT)
       {
          _zustandLehne = LehneZustand.LEHNE_ZURUECK;
          _display.neueNachricht(Nachrichten.ZURUECK_UNTEN_AUS);
       }
    }

    public void stelleLehneVor()
    {
       if (_zustandLehne == LehneZustand.LEHNE_ZURUECK
             && _zustandMassage == MassageZustand.MASSAGE_AUS)
       {
          _zustandLehne = LehneZustand.LEHNE_AUFRECHT;
          _display.neueNachricht(Nachrichten.AUFRECHT_UNTEN_AUS);
       }
    }

    public void schalteMassageEin()
    {
       if (_zustandLehne == LehneZustand.LEHNE_ZURUECK
             && _zustandMassage == MassageZustand.MASSAGE_AUS)
       {
          _zustandMassage = MassageZustand.MASSAGE_AN;
          _display.neueNachricht(Nachrichten.ZURUECK_UNTEN_EIN);
       }
    }

    public void schalteMassageAus()
    {
       if (_zustandMassage == MassageZustand.MASSAGE_AN)
       {
          _zustandMassage = MassageZustand.MASSAGE_AUS;
          _display.neueNachricht(Nachrichten.ZURUECK_UNTEN_AUS);
       }
    }

    public void stelleFuesseHoch()
    {
       // TODO Implementation
    }

    public void stelleFuesseRunter()
```

```
    {
        // TODO Implementation
    }
}
```

3. Aufgabe

Die bestehende Funktionalität des Sessels soll nun dahingehend erweitert werden, dass die Fußstütze mit den entsprechenden zwei Buttons auf der Fernbedienung verstellt werden kann. In Zukunft soll dann die Massage nur noch genau dann aktivierbar sein, wenn die Lehne zurück- und die Füße hochgestellt sind. Das gewünschte Verhalten wird durch ein ergänztes Zustandsdiagramm spezifiziert (Abbildung 4).

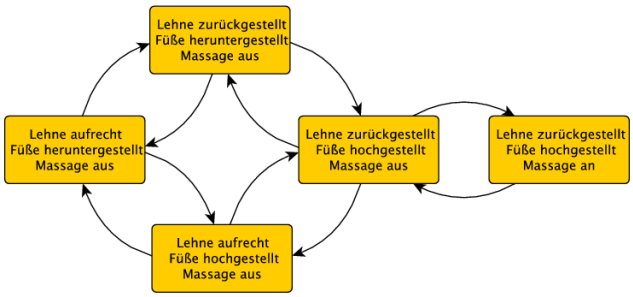


Abbildung 4: Zustandsdiagramm der Zielvorstellung

Beim Blick auf den Code der Klasse `Sessel` wird dann schnell klar, wieso der dort bisher verfolgte Ansatz bei steigender Komplexität immer schwieriger les- und wartbar wird. Es muss ein anderer Weg gefunden werden, die Zustände zu modellieren.

4. Choreographie

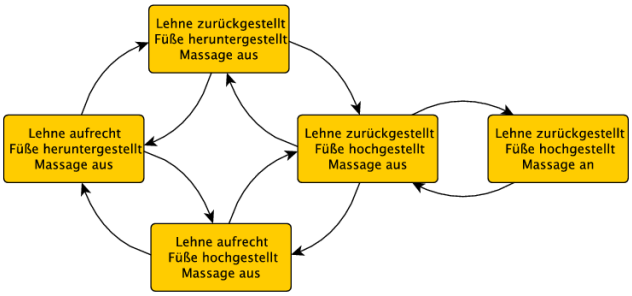
An dieser Stelle beschreiben wir, wie das Teachlet aufgeführt werden kann, in welcher Reihenfolge die einzelnen Schritte passieren und wie lange sie jeweils dauern. Die Zeitangaben basieren auf der ersten Durchführung des Teachlets und sind lediglich Empfehlungen. Wir geben die Zeit in Minuten und kumuliert an, d.h. bei jeder Einheit steht, wie viel Zeit am Ende dieser Einheit insgesamt in etwa vergangen sein sollte.

Es empfiehlt sich, vor Beginn des Teachlets die Folien sowie im Hintergrund Eclipse mit dem Ausgangssystem geöffnet zu haben.

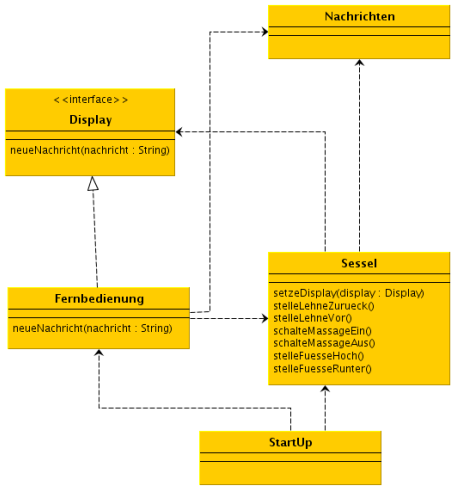
- **Begrüßung** *(beendet nach 2 min)*
  Für Ruhe sorgen, sich vorstellen etc.
- **Vorgeschichte** *(beendet nach 4 min)*
  Um Stimmung zu erzeugen, kann hier die Vorgeschichte motiviert werden: *Enterprise Sessel Solutions ist eine Firma, die Sessel für Enterprise-Kunden herstellt und verkauft. Gerade solche High-Profile-Kunden sind natürlich höchste Qualität gewöhnt, weshalb die Massagesessel der neuesten Reihe mit leistungsfähigen Mikroprozessoren ausgestattet werden, welche in Java programmierbar sind. Das Programmierteam, welches die Sessel programmieren sollte, hat jedoch kürzlich das Handtuch geworfen, weil (Zitat) „das Problem nicht mit vertretbarem Aufwand gelöst werden kann“. Dabei sind längst Verträge unterschrieben und Aufträge angenommen! Das Management von Enterprise Sessel Solutions wendet sich hilfesuchend an euch.*
- **Ausgangssystem in Benutzung erkunden** *(beendet nach 10 min)*
  Hier können die Teilnehmer das Ausgangssystem kennenlernen. Da es nicht allzu viele Interaktionsmöglichkeiten gibt, versuchen sie hoffentlich schon bald, das Verhalten des Sessels zu durchschauen. Es kann darauf hingeleitet werden, sich um Zustände und Zustandsübergänge Gedanken zu machen.

- **Zustandsdiagramm erstellen** *(beendet nach 13 min)*
  Idealerweise sind die Teilnehmer mit Zustandsdiagrammen grundlegend vertraut. In diesem Teachlet werden sie allerdings nicht allzu anspruchsvoll. Wir verzichten bewusst auf Kantenbeschriftungen um Zeit zu sparen, da die Übergänge in diesem System intuitiv schnell erfassbar sind.
  Wenn das Zustandsdiagramm an einer zuklappbaren Tafel erstellt wird, bietet es sich an, bereits vor Beginn des Teachlets einen Zustand anzuzeichnen, um den Teilnehmern ein Beispiel vorzulegen und die Erstellung etwas abzukürzen. Das Zustandsdiagramm wird in den Folien noch einmal wiederholt.
  Ergebnis dieser Einheit:

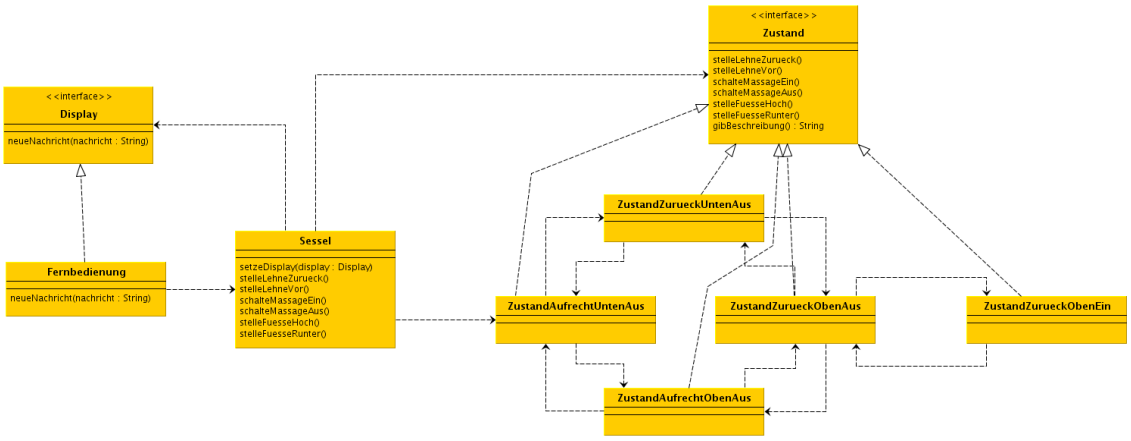
- **Arbeitsauftrag motivieren** *(beendet nach 15 min)*
  Noch vor dem ersten Blick in den Code gibt es nun den Arbeitsauftrag: Die Funktionalität des Sessels soll um die Fußstütze erweitert werden. Die Spezifikation wird textuell gegeben und die Teilnehmer sollen das vorhandene Zustandsdiagramm erweitern. Allen Teilnehmern sollte klar sein, dass Lehne und Füße unabhängig voneinander verstellt werden können, die Massage aber nur aus einer Position heraus aktiviert werden kann.
- **Zustandsdiagramm erweitern** *(beendet nach 20 min)*
  In diesem Schritt wird interaktiv das erweiterte Zustandsdiagramm an die Tafel gebracht. Dieses wird in den Folien noch einmal wiederholt.
  Nach dieser Einheit sollte das Handout verteilt werden, da es einen Screenshot des Systems sowie die zwei Zustandsdiagramme enthält, also genau die Informationen, die die Teilnehmer ab diesem Punkt stets vor Augen haben sollten.
  Ergebnis dieser Einheit:



- **Code-Inspektion** *(beendet nach 35 min)*
  Hier können die Teilnehmer erstmals den Code des Ausgangssystems erkunden. Falls keine Anregungen aus dem Plenum kommen, bietet es sich an, die Klassen in der Reihenfolge `StartUp`, `Fernbedienung`, `Display`, `Sessel`, `Nachrichten` zu betrachten. Die Methode `initGui` in der Fernbedienung muss nicht genau analysiert werden, die `ActionListener` an den sechs Buttons sind allerdings relevant.
- **Klassendiagramm des Ausgangssystems erstellen** *(beendet nach 45 min)*
  Im Anschluss an die Code-Inspektion wird gemeinsam ein Klassendiagramm des Ausgangssystems erstellt. Die Klassen `StartUp` und `Nachrichten` sind weitgehend irrelevant und können problemlos weggelassen werden, ebenso muss die Schnittstelle der Klassen nicht detailliert wiedergegeben werden.
  Achtung: Das Klassendiagramm ist in den Folien nicht reproduziert.
  Ergebnis dieser Einheit:



- **Diskussion des Problems, Lösungsansätze finden** *(beendet nach 55 min)*
  An diesem Punkt kann eine freie Diskussion gestartet werden, wie die Problemstellung gelöst werden kann. Mit viel Glück kommen die Teilnehmer selbst auf das Zustandsmuster, ansonsten kann die Diskussion auch in die Richtung gelenkt werden. Falls entweder gar nichts oder zu viel Unterschiedliches kommt, kann die Diskussion jederzeit abgebrochen werden. *(Vorschlag: Der Praktikant kommt hereingestürmt, er hat gegooglet und dabei eine Beschreibung des Zustandsmusters gefunden.)*
- **Vorstellung des Zustandsmusters** *(beendet nach 65 min)*
  Egal ob die Teilnehmer von sich aus darauf gekommen sind oder nicht: An dieser Stelle erhalten sie eine Einführung in das Zustandsmuster wie die „Gang of four“ es beschrieben hat (Gamma et al., 1995). Zusätzlich zur originalen Beschreibung gibt es eine Reihe neuerer Literatur, wir empfehlen Freeman et al. (2004) aufgrund der pragmatischen und anschaulichen Beschreibungsweise sowie der Erklärung anhand von Java-Code.
- **Klassendiagramm erweitern** *(beendet nach 75 min)*
  Das bestehende Klassendiagramm soll nun erweitert werden, um einen Entwurf zu planen, der das Zustandsmuster umsetzt. Unser Zielsystem gibt den Typ `Zustand` als Interface an, die häufigste Variation wird vermutlich eine abstrakte Klasse sein. Auf die Idee, die Zustandsübergänge zu realisieren indem die konkreten Zustände den Nachfolgezustand jeweils zurückgeben, müssen die Teilnehmer ggf. gebracht werden.
  Auch dieses erweiterte Klassendiagramm findet sich nicht in den Folien.
  Ergebnis dieser Einheit:



- **Umsetzung** *(beendet nach 90 min)*
  Nun soll der Entwurf in Eclipse umgesetzt werden. Im Kapitel „Zielsystem“ findet sich eine ausführliche Beschreibung, wie das geschehen kann. Falls genug Impulse von den Teilnehmern kommen, sollten diese umgesetzt werden, statt sich strikt an das vorgegebene Zielsystem zu halten. Falls die Teilnehmer keinen zufriedenstellenden Entwurf erstellen konnten, kann stattdessen das vorhandene Zielsystem gezeigt

werden.

- **Zusammenfassung** *(beendet nach 95 min)*
  Es folgt eine Zusammenfassung der Lösung mit erneuter Hervorhebung der Vorteile des Zustandsmusters. Die Teilnehmer werden zur gelungenen Umsetzung beglückwünscht.
- **Diskussion über Alternativen** *(beendet nach 103 min)*
  Sofern noch Zeit und Energie vorhanden ist, kann an dieser Stelle über Varianten des Zustandsmusters oder alternative Entwürfe, die vorher verworfen wurden, diskutiert werden.
- **Ende** *(beendet nach 105 min)*
  Das Teachlet ist beendet.

5. Zielsystem

In diesem Abschnitt wollen wir das zu diesem Teachlet gehörende Zielsystem beschreiben und erläutern, wie dieser oder ein vergleichbarer Entwurf im Plenum sinnvoll umgesetzt werden kann.

5.1. Beschreibung

Das vorliegende Zielsystem ist die Implementation einer möglichen und unserer Meinung nach in dieser Situation angemessenen Variante des Zustandsmusters. Bei der Durchführung des Teachlets kann es passieren, dass die Teilnehmer eine Variante konzipieren, die sich von dieser unterscheidet. Zum Vortragen des Teachlets ist deshalb dringend dazu zu raten, sich im Detail mit den verschiedenen Varianten des Musters auseinanderzusetzen und ggf. zur Übung einige weitere zu implementieren, etwa mit einer abstrakten Klasse `Zustand` statt eines Interfaces.

Dieses Zielsystem sieht als Klassendiagramm anders aus als das Ausgangssystem, da für die Zustände des Sessels neue Klassen erzeugt wurden (Abbildung 5). Auffallend und unserer Meinung nach besonders interessant ist, dass die Zustände und ihre „benutzt“-Beziehungen in dem Klassendiagramm genau dem spezifizierenden Zustandsdiagramm entsprechen.

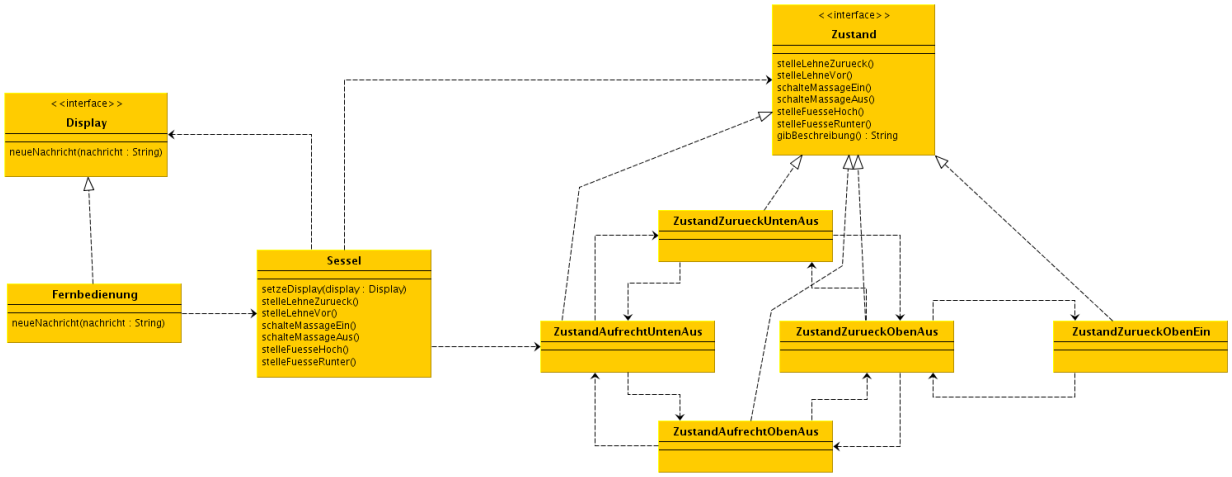


Abbildung 5: Klassendiagramm des Zielsystems; aufgrund der Übersichtlichkeit wurden `StartUp` und `Nachrichten` weggelassen

Aufmerksame Teilnehmer werden feststellen, dass dieser Entwurf zwischen den Zuständen zyklische Abhängigkeiten enthält. Unserer Meinung nach sind diese an dieser Stelle jedoch nicht dogmatisch abzulehnen, da zyklische Abhängigkeiten ein Zeichen für zu enge Kopplung zwischen Klassen sein können, aber nicht müssen. In diesem Fall benutzen sich die Zustandsklassen nur genau für die Zustandsübergänge und die strukturelle Ähnlichkeit von Zustands- und Klassendiagramm ist ein durchaus angenehmer Effekt.

Falls die Gruppe innerhalb der Zeit nicht auf eine zufriedenstellende Lösung kommt, kann dieses Zielsystem gezeigt werden, um die praktische Umsetzung des Musters zu zeigen.

5.2. Änderungen

Die nötigen Änderungen am Ausgangssystem lassen sich wie folgt zusammenfassen:

1. **Neues Interface `Zustand`**: Ein Interface `Zustand` muss erzeugt werden, welches von den konkreten Zuständen implementiert wird.
2. **Konkrete Zustände**: Die fünf Zustände müssen implementiert werden, und zwar als Klassen, die das Interface `Zustand` implementieren und bei Zustandswechseln jeweils ein Exemplar eines anderen konkreten Zustandes zurückgeben.
3. **Anpassen des Sessels**: In der Klasse `Sessel` muss die komplette Zustandslogik einschließlich der `enum`s entfernt werden. Stattdessen muss ein Zustand erzeugt und gehalten werden, an den in den Methoden des Sessels das Zustandsverhalten delegiert wird.

Es folgt nun eine ausführliche Beschreibung, wie diese Änderungen mit der verwendeten Entwicklungsumgebung (Eclipse 3.4) durchgeführt werden können.

Als erstes muss das Interface `Zustand` erstellt werden. Dazu bietet es sich an, die passenden Operationen mit dem Refactoring „Extract Interface“ aus der Klasse `Sessel` zu extrahieren. Dazu: Rechtsklick auf die Klasse `Sessel`, „Refactor“, „Extract Interface“; in der Dialogbox die Häkchen für alle sechs Methoden außer `setzeDisplay` setzen sowie das Häkchen bei „Declare interface methods as 'abstract'“ entfernen, dann auf „OK“ klicken. Im neu entstandenen Interface alle Rückgabetypen von `void` auf `Zustand` ändern. Danach muss in der Klasse `Sessel` noch das „`implements Zustand`“ entfernt werden. Außerdem muss zum `Zustand` noch die Operation `gibBeschreibung` mit Rückgabetyp `String` hinzugefügt werden.

Das Ergebnis dieses Arbeitsschrittes ist das fertige Interface `Zustand` (Listing 2).

Listing 2: Zustand.java

```
1  public interface Zustand
   {
      public Zustand stelleLehneVor();

5     public Zustand stelleLehneZurueck();

      public Zustand schalteMassageEin();

      public Zustand schalteMassageAus();
10
      public Zustand stelleFuesseHoch();

      public Zustand stelleFuesseRunter();

15    public String gibBeschreibung();
   }
```

Als zweites können die fünf Klassen implementiert werden, welche die fünf Zustände modellieren, die der Sessel einnehmen kann. Dazu eine neue Klasse erzeugen, die das Interface `Zustand` implementiert (Eclipse kann dann automatisch die entsprechenden Methoden leer anlegen). Bei der Benennung der Zustandsklassen kann sich an den Konstanten in der Klasse `Nachrichten` orientiert werden, so erhält man fünf Klassen mit Namen wie `ZustandAufrechtUntenAus`. Die Methodenrümpfe in diesen Klassen enthalten dann lediglich Einzeiler wie „`return new ZustandAufrechtObenAus();`“ bei Aktionen, in denen der Zustand gewechselt werden soll, und „`return this;`“ wenn bei einer Aktion kein Zustandswechsel erfolgen soll. Die Ausnahme dazu ist `gibBeschreibung`, in der die passende Konstante aus `Nachrichten` zurückgegeben wird.

Als Beispiel für die Implementation eines konkreten Zustandes geben wir an dieser Stelle den Quelltext der Klasse `ZustandAufrechtUntenAus` aus dem Zielsystem wieder (Listing 3).

Listing 3: ZustandAufrechtUntenAus.java

```
public class ZustandAufrechtUntenAus implements Zustand
{
   public Zustand stelleLehneVor()
   {
      return this;
   }

   public Zustand stelleLehneZurueck()
   {
      return new ZustandZurueckUntenAus();
   }

   public Zustand schalteMassageEin()
   {
      return this;
   }

   public Zustand schalteMassageAus()
   {
      return this;
   }

   public Zustand stelleFuesseHoch()
   {
      return new ZustandAufrechtObenAus();
   }

   public Zustand stelleFuesseRunter()
   {
      return this;
   }

   public String gibBeschreibung()
   {
      return Nachrichten.AUFRECHT_UNTEN_AUS;
   }
}
```

Als drittes muss der Sessel angepasst werden. Dazu aus `Sessel` die `enum`s und sämtliche Zustandsübergangslogik löschen. Stattdessen eine neue private Exemplarvariable `_zustand` vom Typ `Zustand` einbauen, der im Konstruktor mit „`_zustand = new ZustandAufrechtUntenAus();`“ der Startzustand zugewiesen wird. In den sechs verbleibenden Methoden müssen dann noch jeweils zwei Dinge geschehen: Die gewünschte Aktion muss an den aktuellen Zustand delegiert werden, wobei bei einem Zustandswechsel der Rückgabewert dem neuen Zustand entspricht, also z.B. „`_zustand = _zustand.stelleLehneVor();`“. Außerdem muss danach in jeder der sechs Methoden noch die Statusanzeige aktualisiert werden, dies geschieht durch die folgende Anweisung: „`_display.neueNachricht(_zustand.gibBeschreibung());`“ - Dies passiert also immer und unabhängig davon, ob tatsächlich eine Veränderung des Zustands stattgefunden hat.

Auch die angepasste Klasse `Sessel` wollen wir an dieser Stelle wiedergeben (Listing 4).

Listing 4: Sessel.java

```
public class Sessel
{
   private Zustand _zustand;
   private Display _display;

   public Sessel()
   {
      _zustand = new ZustandAufrechtUntenAus();
   }

   public void setzeDisplay(Display display)
   {
      _display = display;
```

```
    }

    public void stelleLehneZurueck()
    {
        _zustand = _zustand.stelleLehneZurueck();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }

    public void stelleLehneVor()
    {
        _zustand = _zustand.stelleLehneVor();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }

    public void schalteMassageEin()
    {
        _zustand = _zustand.schalteMassageEin();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }

    public void schalteMassageAus()
    {
        _zustand = _zustand.schalteMassageAus();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }

    public void stelleFuesseHoch()
    {
        _zustand = _zustand.stelleFuesseHoch();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }

    public void stelleFuesseRunter()
    {
        _zustand = _zustand.stelleFuesseRunter();
        _display.neueNachricht(_zustand.gibBeschreibung());
    }
}
```

Damit sollte das Ausgangssystem erfolgreich in das Zielsystem umgebaut worden sein.

6. Bisherige Einsätze

Es folgt eine Auflistung der bisherigen Einsätze des Teachlets einschließlich Bewertung des Plenums (sofern erhoben).

6.1. Teachlet-Werkstatt 2009

Dieses Teachlet wurde entwickelt für und erstmalig durchgeführt im Seminar „Konzepte objektorientierter Programmiersprachen“ am Department Informatik der Universität Hamburg im Sommersemester 2009. Die Erstaufführung (Teachlet-Version 1.0.0) geschah am 09.06.2009 mit Julian Fietkau und Janina Nemec als Moderatoren.

Die durchschnittliche Bewertung des Plenums (16 Stimmen) auf einer Skala von 0 (schlecht) bis 5 (optimal):

- Ausgangssystem: 4,0
- Foliensatz: 4,5
- Eignung für das Lernziel: 4,5

Bisher ist es bei dieser einen Aufführung geblieben.

7. Offene Punkte

7.1. Große Änderungen

- Das Teachlet behandelt zur Zeit lediglich die Variante des Zustandsmusters mit dezentraler Übergangsverwaltung, d.h. jeder konkrete Zustand kennt seine möglichen Nachfolgezustände. Eine weitere verbreitete Variante ist die zentrale Übergangsverwaltung, in der der Kontext (der Sessel) die Logik für die Zustandsübergänge enthält, die einzelnen Zustände dagegen nur das jeweils zustandsspezifische Verhalten. Das Teachlet ist jedoch auch mit dieser Beschränkung zeitlich bereits recht lang, so dass andere Varianten des Musters je nach verbleibender Zeit am Ende des Teachlets kurz diskutiert werden können. Falls dafür keine Zeit mehr ist, muss darauf verzichtet werden.
- Eventuell lohnt es sich, das Ausgangssystem visueller zu gestalten, indem bspw. die Statustexte durch Piktogramme eines Sessels ersetzt werden. Dies könnte die Attraktivität des Systems steigern und dazu führen, dass die Funktionalität schneller verstanden wird; das System wäre sozusagen spannender. Allerdings wäre dies mit einer erhöhten Komplexität der Software erkauft und die Bilder müssten erstellt werden.
- Die verwendeten Bilder wurden ohne Rücksicht auf evtl. vorhandene Urheberrechtsansprüche ausgewählt und verwendet. Um eine Veröffentlichung der Unterlagen zu diesem Teachlet unproblematischer zu machen, könnten die urheberrechtlich geschützten und nicht für dieses Teachlet erstellten Bilder durch gemeinfreie, frei lizensierte oder eigene Bilder ersetzt werden. Es handelt sich bei den problematischen Bildern um das Foto des Sessels und das der jubelnden Leute. Das Lupen-ClipArt ist bereits gemeinfrei.

7.2. Kleine Änderungen

- Momentan findet das Binden der sechs Buttons an die Schnittstelle des Sessels bei der Erzeugung der ActionListener innerhalb der `initGui`-Methode der Klasse `Fernbedienung` statt. Das ist etwas unschön, weil auf diese Weise für das Verständnis wichtige Codezeilen in einer Methode stehen, die eigentlich als für das Lernziel irrelevant gelten sollte. Vielleicht findet sich eine sinnvolle Möglichkeit, diesen Code anders zu strukturieren, um die GUI-Initialisierung und die Aufrufe am Sessel sauberer zu trennen.
- Das Zustandsmuster ist strukturell gesehen identisch mit dem Strategiemuster, lediglich die Einsatzzwecke unterscheiden sich. Wird dieses Teachlet vor einer Gruppe gehalten, die das Strategiemuster kennt, bietet es sich an, die Gemeinsamkeiten und Unterschiede an geeigneter Stelle zu erläutern.
- Ein Teilnehmer hat bemängelt, dass die Spiegelstriche auf den Folien einzeln eingeblendet werden. Wir halten das für eine Geschmacksfrage. Falls gewünscht, kann für eine zukünftige Vorführung dieses Teachlets dieses Verhalten durch das Aus- bzw. Einkommentieren einer Zeile im LaTeX-Code der Folien geändert werden.

8. Literatur

- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1995
- Freeman, E.; Freeman, E.; Bates, B.; Sierra, K.: Head First Design Patterns. O'Reilly Media, 2004