Übungen zur Vorlesung

Softwaretechnologie

- Wintersemester 2011/2012 -

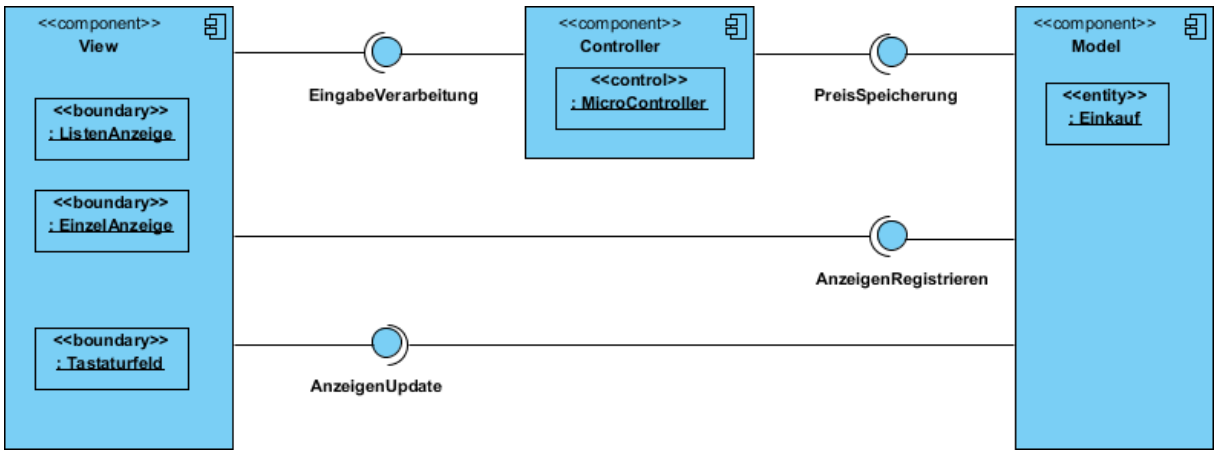
Dr. Günter Kniesel

Übungsblatt 9 - Lösungshilfe

Aufgabe 1. *Systementwurf* (7 Punkte)

Die Softwarefirma entscheidet sich, eine **Model-View-Controller-Architektur** für die Modellierung der Kasse zu verwenden, bei dem sich die Anzeigen beim Initialisieren am Modell zwecks Aktualisierungs-Benachrichtigung anmelden.

a) Überlegen Sie sich eine sinnvolle Gruppierung der Analyseklassen in Komponenten (Subsysteme) und definieren Sie Dienste. Zeichnen Sie ein passendes **Komponentendiagramm.**



b) Diskutieren Sie, wie die Wahl für die **Model-View-Controller-Architektur** folgende Entwurfsziele erfüllt oder verletzt:

- Erweiterbarkeit (z.B. neue „Views“)
  - Sehr gut, solange die Domäne nicht erweitert wird
  - Neue Views: problemlose Anmeldung
  - Neuer Controller: problemlose Einbindung
  - Erweiterung der Anwendungsdomäne
    - Erweiterung des model-Subsystems
    - Zur geeigneten Darstellung des neuen Wissens, müssten evtl. die bestehenden view-Subsysteme angepasst oder neue erzeugt werden.
    - Evtl. Anpassung Der Interaktionen zwischen Benutzer und model-Subsystem auf die neuen Daten
      - neue controller-Subsysteme hinzufügen bzw.
      - bestehende verändern.

- Reaktionszeit (Zeit zwischen einer Benutzereingabe und dem Abschluss der Aktualisierung aller Views)
  - Abhängig von System und Anforderungen
  - Für Echtzeitsysteme evtl. nicht sinnvoll
    - Zugriff auf die Domänendaten erfordert Zwischenschritte
  - Schlecht, wenn sich der Zustand des model-Subsystems oft ändert
    - viel Kommunikation zwischen den Sub-Systemen
  - Höhere Effizienz z.B. durch Zusammenfassen mehrerer Nachrichten zu einer
- Änderbarkeit (z.B. die Erweiterung des Modells um zusätzliche Attribute)
  - Sehr gut, solange sich die Domäne nicht ändert
  - Analog zum Entwurfsziel „Erweiterbarkeit“
  - Frage auch hier: Muss zur Erweiterung des Models (neue Attribute oder Operationen) sein Interface geändert werden?
- Zugriffs-Kontrolle (d.h. die Sicherstellung, dass nur berechtigte Benutzer auf bestimmte Teile des Modells zugreifen können)
  - Relativ gut erfüllt
  - Verantwortung für Aktionen auf Model liegt bei den Controllers
    - Controller sollte nur mit vertrauenswürdigen Nutzern interagieren
  - Verantwortung für Visualisierung des Models liegt bei Views
    - Falls nicht alle Änderungen visualisiert werden sollen kann man einen Controller zwischen Model und Views stellen
    - Sieht für Model wie ein View aus
    - Datenweitergabe nur an vertrauenswürdige Views

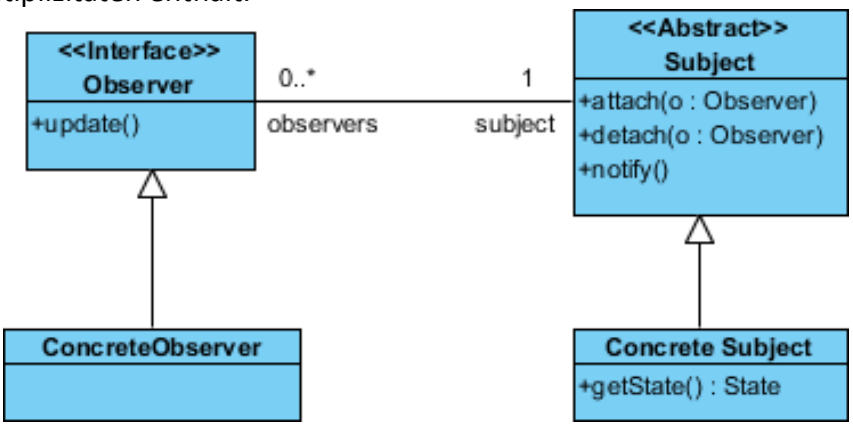
Aufgabe 2. *Entwurfsmuster* (4 Punkte)

In Aufgabe 2 sollten Sie auf eine zyklische Abhängigkeit zwischen der View- und der Model-Komponente gestoßen sein.

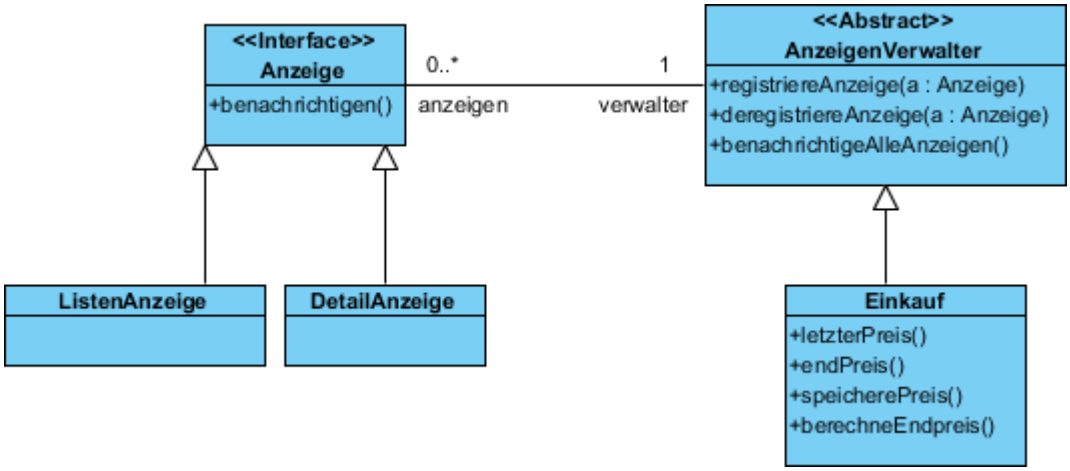
a) Mit welchem Entwurfsmuster können Sie – ohne die Kommunikation zu ändern – eine daraus folgende Abhängigkeit auf Klassenebene vermeiden?

- Observer-Pattern (Beobachter)

b) Wie ist dieses Pattern **im Allgemeinen** aufgebaut? Zeichnen Sie ein entsprechendes Klassendiagramm, das die wichtigsten Klassen und Methoden sowie Assoziationen und Multiplizitäten enthält.



c) Wie lässt sich das Pattern **im konkreten Fall der Situation aus Aufgabe 1** einsetzen? Zeichnen Sie, analog zu Teilaufgabe b, ein Klassendiagram und halten Sie die Bezeichnungen konsistent zu Ihren Lösungen in Aufgabe 1.



Aufgabe 3. *Entwurfsmuster* (6 Punkte)

Für die Verwaltung der gemeinsamen Termine aller Studierenden hat die Fachschaft mit der Entwicklung einer „Appointment-Anwendung“ begonnen. Das Programm soll Appointments (Termine), die aus einer Beschreibung und einem Datum bestehen, verwalten. Eine erste Version dieses Programms finden Sie im SVN-Repository im Ordner „share“ als „AppointmentManager.zip“

Inzwischen haben sich so viele Termine angesammelt, dass auch eine Gruppierung der Termine unterstützt werden soll.

c) Überarbeiten Sie das Programm mit Hilfe eines Entwurfsmusters. Es soll

a. möglichst wenig an den bestehenden Klassen geändert werden. Vor allem soll die Funktionalität der Klasse *AppointmentItem* erhalten bleiben.

b. die Möglichkeit bestehen, Gruppen, die wiederum (Unter-)Gruppen oder natürlich auch *Appointment*-Einträge enthalten, einzufügen.

c. möglich sein, überall Gruppen anzutreffen, wo *Appointment*-Einträge erwartet werden und umgekehrt. Insbesondere muss eine Gruppe dieselben Methoden unterstützen, die auch ein einfacher Eintrag bietet.

```
public interface Appointment {
	public String getDescription();
	public void setDescription(String description);
	public String toString();

	public void add(Appointment a);
	public void remove(Appointment a);
	public Set<Appointment> getChildren();
}

public class AppointmentGroup implements Appointment {
```

```
	private String description;
	private Set<Appointment> children;

	public AppointmentGroup(String description){
		children = new HashSet<Appointment>();
		this.description = description;
	}

	@Override
	public String getDescription() {
		return description;
	}

	@Override
	public void setDescription(String description) {
		this.description = description;
	}

	@Override
	public void add(Appointment a) {
		children.add(a);
	}
	@Override
	public void remove(Appointment a) {
		children.remove(a);
	}
	@Override
	public Set<Appointment> getChildren() {
		return children;
	}


	@Override
	public String toString() {
		StringBuffer result = new StringBuffer();
		result.append("AppointmentGroup *"+description+"* : {\n");

		for (Appointment a: children) {
			result.append(a.toString());
		}

		result.append("}\n");

		return result.toString();
	}

}
```

d) In der Klasse *AppointmentManagerTest* findet sich ein erstes Programm, das ein paar Beispieleinträge erzeugt und dann auf die Konsole ausgibt. Stellen Sie sicher, dass nach Ihrer Anpassung des Models die Ausgabe noch funktioniert. Kopieren Sie die Klasse und passen Sie die Kopie so an, dass die Einträge in Gruppen geordnet werden. Es sollen zusätzlich zu den einfachen Einträgen auch die Gruppeneinträge und ihre Unterelemente ausgegeben werden, ohne dass der entsprechende *println*-Aufruf oder die *AppointmentManager*-Klasse geändert werden.

```
public class AppointmentManagerGroupedTest {
```

```
        static DateFormat format =
                  new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        public static void main(String[] args) throws ParseException {
              AppointmentManager manager = new AppointmentManager();

              Appointment mittwoch = new AppointmentGroup("Mittwoch");
              manager.addItem(mittwoch);

              mittwoch.add(new AppointmentItem("Ordnerschulung ",
                                        format.parse("2009-12-09 16:00:00")));
              mittwoch.add(new AppointmentItem("Bildungsgebet",
                                        format.parse("2009-12-09 18:00:00")));


              Appointment donnerstag = new AppointmentGroup("Donnerstag");
              manager.addItem(donnerstag);

              Appointment vormittags = new AppointmentGroup("Vormittags");
              donnerstag.add(vormittags);
              vormittags.add(new AppointmentItem("Treffen zur KMK-Demo",
                                        format.parse("2009-12-10 09:30:00")));
              vormittags.add(new AppointmentItem("Treffen am Bonner Hbf",
                                        format.parse("2009-12-10 12:00:00")));

              Appointment nachmittags = new AppointmentGroup("Nachmittags");
              donnerstag.add(nachmittags);
              nachmittags.add(new AppointmentItem("Großdemo”",
                                        format.parse("2009-12-10 13:00:00")));
              nachmittags.add(new AppointmentItem("Großes Plenum",
                                        format.parse("2009-12-10 18:00:00")));


              Appointment freitag = new AppointmentGroup("Freitag");
              manager.addItem(freitag);

              freitag.add(new AppointmentItem("KMK-Nachbereitung in HS1",
                                        format.parse("2009-12-11 14:00:00")));
              freitag.add(new AppointmentItem("Plenum",
                                        format.parse("2009-12-11 18:00:00")));

              System.out.println(manager);
        }
}
```