Übungen zur Vorlesung

Softwaretechnologie

- Wintersemester 2010/11 -

Dr. Günter Kniesel

Übungsblatt 7 - Lösungshilfe

Aufgabe 1. *Entwurfsmuster* (12 Punkte)

Gegeben sei das Bibliotheksverwaltungs-Projekt, das bereits im letzten Übungsblatt vorgestellt wurde. Der Programmcode steht im SVN-Repository, im Ordner „share“, als „Bibliothek.zip“ zur Verfügung.

Das existierende System sieht für Ausleihgegenstände nur eine Klasse vor. Diese hat ein Feld, welches den Typ des Mediums angibt. Diese Entscheidung ist sinnvoll, wenn man davon ausgeht, dass sich die Medien vom Verhalten nicht sehr unterscheiden.

Unser System sollen aber in Zukunft an Dritte verkauft werden und ihnen die Möglichkeit bieten, in ihre Variante der Bibliothek neue Medienarten mit eigenen Verhalten zu integrieren, ohne die von uns erstellten Klassen ändern zu müssen. Beispielsweise sollen Videos innerhalb einer Ausleihe dem Benutzer online abgespielt werden können.

Das bisherige Design soll daher nun modifiziert werden.

a) Welches Entwurfsmuster bietet sich an, um beliebige Medien-Objekte mit dem statischen Typ *Ausleihgegenstand* erzeugen, ohne dass der konkrete Typ der möglichen Objekte schon bei der Entwicklung der Bibliothek bekannt sind?

Factory Method

b) Welche Rollen gibt es in diesem Entwurfsmuster? Wie können Sie diese Rolle bestehenden oder noch zu entwickelnden Klassen des Projektes zuordnen?

Creator (Abstrakt) → *neu* AbstrakterMedienerzeuger (*oder* Bibliothek)
Product (Abstrakt) → Ausleihgegenstand
ConcreteCreator → *neu* Medienerzeuger *oder* Bibliothek
ConcreteProduct → *neu* Video, Buch, Zeitschrift etc.

c) Implementieren Sie das Entwurfsmuster im Projekt. Entwerfen Sie dabei Klassen für *Video*, *Buch* und *Zeitschrift*, die von der Klasse *Ausleihgegenstand* erben und integrieren Sie diese Klassen in Ihr Projekt (ohne Änderungen an *Ausleihgegenstand* und nur minimale Änderung[1] an *Bibliothek*).

```
public abstract class AbstrakterMedienerzeuger {

   protected abstract Ausleihgegenstand erzeugeAusleihgegenstand
                          (String id, String typ, String titel);
```

[1] Z.B. ein neues Feld und einen modifizierten Methodenaufruf.

```
}

public class Video extends Ausleihgegenstand {
   Video(String nomenklatur, String titel) {
         super(nomenklatur, "Video", titel);
   }

}

public class Buch extends Ausleihgegenstand {
   Buch(String nomenklatur, String titel) {
         super(nomenklatur, "Buch", titel);
   }
}

public class Zeitschrift extends Ausleihgegenstand {
   Zeitschrift(String nomenklatur, String titel) {
         super(nomenklatur, "Zeitschrift", titel);
   }

}

public class Medienerzeuger extends AbstrakterMedienerzeuger {

   @Override
   protected Ausleihgegenstand erzeugeAusleihgegenstand
                        (String id, String typ, String titel) {

         // Bekannte Medien mit spezifischem Typ
         if ("Video".equals(typ)) {
               return new Video(id, titel);
         } else if ("Buch".equals(typ)) {
               return new Buch(id, titel);
         } else if ("Zeitschrift".equals(typ)) {
               return new Zeitschrift(id, titel);
         }

         // Unbekannte Medien mit generischem Typ
         return new Ausleihgegenstand(id, typ, titel);
   }

}

public class Bibliothek {
   …
   protected AbstrakterMedienerzeuger medienerzeuger =
                                          new Medienerzeuger();
   …
   public Ausleihgegenstand neuerAusleihgegenstand
                        (String id, String typ, String titel) {
         Ausleihgegenstand neuerGegenstand =
               medienerzeuger.erzeugeAusleihgegenstand(id, typ, titel);
         …
   }


}
```

d) Erstellen Sie ein neues Java-Projekt „Unibibliothek“ und fügen Sie das Projekt „Bibliothek“ in den Build-Path des neuen Projektes ein. Entwickeln Sie hier nun mit minimalem Aufwand auf Basis von Aufgabenteil (c) eine *Unibibliothek*, die zusätzlich die Typen *Seminararbeit* und *Uralter Wälzer* kennt.

```
public class Seminararbeit extends Ausleihgegenstand {
	Seminararbeit(String nomenklatur, String titel) {
		super(nomenklatur, "Seminararbeit", titel);
	}
}

public class UralterWälzer extends Ausleihgegenstand {
	UralterWälzer(String nomenklatur, String titel) {
		super(nomenklatur, "Uralter Wälzer", titel);
	}
}

public class UniMedienerzeuger extends Medienerzeuger {

	@Override
	protected Ausleihgegenstand erzeugeAusleihgegenstand
			(String id, String typ, String titel) {

		if ("Seminararbeit".equals(typ)) {
			return new Seminararbeit(id, titel);
		} else if ("Uralter Wälzer".equals(typ)) {
			return new UralterWälzer(id, titel);
		}
		return super.erzeugeAusleihgegenstand(id, typ, titel);
	}

}

public class Unibibliothek extends Bibliothek {
	public Unibibliothek() {
		this.medienerzeuger = new UniMedienerzeuger();
	}
}
```

Aufgabe 2. *Entwurfsmuster* (12 Punkte)

Passend zur Vorweihnachtszeit hatte ein großer Discounter in der vergangenen Woche verschiedene Funksteckdosen verschiedenen Hersteller im Angebot: Einfache Dosen, dimmbare Dosen und wetterfeste Dosen. Alle Steckdosen können mit einer Fernbedienung gesteuert werden.

Die Fernbedienung enthält u.A. vier frei belegbare Reihen von An-/Aus-Knöpfen, sowie eine Reihe „Master An / Master Aus“, mit der alle angeschlossenen Steckdosen ein- oder ausgeschaltet werden.
Leider konnte die Steuerungs-Entwicklung nicht rechtzeitig abgeschlossen werden. Den bisherigen Code finden Sie im SVN-Repository, im Ordner „share“ als „Fernbedienung.zip“.

a) Mit welchem Entwurfsmuster ließe sich die folgende Funktionalitäten der Fernbedienung realisieren:

   a. Die An/Aus Knöpfe sollen paarweise einem beliebigen Gerät zugeordnet werden können.

   b. Alle Dosenarten sollen unterstützt werden, ohne die Klassen zu verändern, welche die einzelnen Gerätearten darstellen.

   c. Neue Geräte sollen später hinzugefügt werden können, ohne existierende Klassen verändern zu müssen.

   Unten ausgeführt: Command-Pattern (Alternativ: Strategy. Definitiv kein State-Pattern, das ja die Modellierung von Zustandsübergängen zentral beinhaltet. Ganz abgesehen davon, sind weder State noch Strategy bisher besprochen worden! ;-)

b) Implementieren Sie die benötigten Klassen und realisieren sie das Entwurfsmuster in der Steuerung der Fernbedienung.

```
public interface Command {
	public void execute();
}

public class EinfacheSteckdoseOnCommand implements Command {

	EinfacheSteckdose dose;

	public EinfacheSteckdoseOnCommand(EinfacheSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.powerOn();
	}

}

public class EinfacheSteckdoseOffCommand implements Command {

	EinfacheSteckdose dose;

	public EinfacheSteckdoseOffCommand(EinfacheSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.powerOff();
	}

}
```

```
public class DimmbareSteckdoseOnCommand implements Command {

	DimmbareSteckdose dose;

	public DimmbareSteckdoseOnCommand(DimmbareSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.setzeDimmwert(100);
	}

}


public class DimmbareSteckdoseOffCommand implements Command {

	DimmbareSteckdose dose;

	public DimmbareSteckdoseOffCommand(DimmbareSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.setzeDimmwert(0);
	}

}


public class WetterfesteSteckdoseOnCommand implements Command {

	WetterfesteSteckdose dose;

	public WetterfesteSteckdoseOnCommand(WetterfesteSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.schalteStromEin();
	}

}


public class WetterfesteSteckdoseOffCommand implements Command {

	WetterfesteSteckdose dose;

	public WetterfesteSteckdoseOffCommand(WetterfesteSteckdose dose) {
		super();
		this.dose = dose;
	}

	@Override
	public void execute() {
		dose.schalteStromAus();
	}
```

```
}

public class Fernbedienung {

	private static final int NUMBER_OF_BUTTONROWS = 4;

	Command[] onCommands = new Command[4];
	Command[] offCommands = new Command[4];

	public Fernbedienung() {
	}
	public void powerOnButtonPushed(int slot){
		Command command = onCommands[slot];
		if (command != null)
			command.execute();
	}

	public void powerOffButtonPushed(int slot){
		Command command = offCommands[slot];
		if (command != null)
			command.execute();
	}

	public void masterPowerOnButtonPushed(){
		for (int i = 0; i < NUMBER_OF_BUTTONROWS; i++)
			powerOnButtonPushed(i);
	}

	public void masterPowerOffButtonPushed(){
		for (int i = 0; i < NUMBER_OF_BUTTONROWS; i++)
			powerOffButtonPushed(i);
	}

	public void setCommands(int slot, Command onCommand,
                                          Command offCommand) {
		if ((slot < 0) || (slot >= NUMBER_OF_BUTTONROWS))
				return;
		onCommands[slot] = onCommand;
		offCommands[slot] = offCommand;
	}
}
```

c) Schreiben Sie einen Programm, das jeder Tasten-Reihe ein Gerät zuordnet und anschließend alle Steckdosen nacheinander einmal ein und ausschaltet und dann die Master-Funktion zum Ein- und Ausschalten verwendet. Komplettieren Sie dazu die Klasse in dem Paket *client*.

```
public class DemoRemoteControl {
	public static void main(String[] args) {
		EinfacheSteckdose dose1 = new EinfacheSteckdose();
		EinfacheSteckdose dose2 = new EinfacheSteckdose();
		DimmbareSteckdose dose3 = new DimmbareSteckdose();
		WetterfesteSteckdose dose4 = new WetterfesteSteckdose();

		Fernbedienung fernbedienung = new Fernbedienung();

		fernbedienung.setCommands(0,
			new EinfacheSteckdoseOnCommand(dose1),
			new EinfacheSteckdoseOffCommand(dose1));
```

```
            fernbedienung.setCommands(1,
                    new EinfacheSteckdoseOnCommand(dose2),
                    new EinfacheSteckdoseOffCommand(dose2));
            fernbedienung.setCommands(2,
                    new DimmbareSteckdoseOnCommand(dose3),
                    new DimmbareSteckdoseOffCommand(dose3));
            fernbedienung.setCommands(3,
                    new WetterfesteSteckdoseOnCommand(dose4),
                    new WetterfesteSteckdoseOffCommand(dose4));

            for (int i = 0; i < 4; i++) {
                  fernbedienung.powerOnButtonPushed(i);
                  fernbedienung.powerOffButtonPushed(i);
            }

            fernbedienung.masterPowerOnButtonPushed();
            fernbedienung.masterPowerOffButtonPushed();

      }
}
```

d) Kopieren Sie das Programm aus Aufgabenteil (c) und vereinfachen Sie es, indem Sie das *Abstract Factory* Pattern bei der Belegung der Knöpfe einsetzen. Sie dürfen dazu auch die Geräte-Klassen erweitern, da üblicherweise Zulieferer bestimmte Spezifikationen des Auftraggebers einhalten müssen.

```
public interface CommandFactory {
      public Command getOnCommand();
      public Command getOffCommand();
}

public class EinfacheSteckdose implements CommandFactory {
      …
      @Override
      public Command getOnCommand() {
            return new EinfacheSteckdoseOnCommand(this);
      }

      @Override
      public Command getOffCommand() {
            return new EinfacheSteckdoseOffCommand(this);
      }
}

public class DimmbareSteckdose implements CommandFactory {
      …
      @Override
      public Command getOnCommand() {
            return new DimmbareSteckdoseOnCommand(this);
      }

      @Override
      public Command getOffCommand() {
            return new DimmbareSteckdoseOffCommand(this);
      }
}

public class WetterfesteSteckdose implements CommandFactory {
      …
      @Override
      public Command getOnCommand() {
```

```
            return new WetterfesteSteckdoseOnCommand(this);
      }

      @Override
      public Command getOffCommand() {
            return new WetterfesteSteckdoseOffCommand(this);
      }
}

public class Fernbedienung {
      …
      public void setDevice(int slot, CommandFactory device) {
            if ((slot < 0) || (slot >= NUMBER_OF_BUTTONROWS))
                  return;
            if (device == null)
                  return;

            onCommands[slot] = device.getOnCommand();
            offCommands[slot] = device.getOffCommand();
      }
}

public class DemoRemoteControlWithFactory {
      public static void main(String[] args) {
            EinfacheSteckdose dose1 = new EinfacheSteckdose();
            EinfacheSteckdose dose2 = new EinfacheSteckdose();
            DimmbareSteckdose dose3 = new DimmbareSteckdose();
            WetterfesteSteckdose dose4 = new WetterfesteSteckdose();

            Fernbedienung fernbedienung = new Fernbedienung();

            fernbedienung.setDevice(0, dose1);
            fernbedienung.setDevice(1, dose2);
            fernbedienung.setDevice(2, dose3);
            fernbedienung.setDevice(3, dose4);

            for (int i = 0; i < 4; i++) {
                  fernbedienung.powerOnButtonPushed(i);
                  fernbedienung.powerOffButtonPushed(i);
            }

            fernbedienung.masterPowerOnButtonPushed();
            fernbedienung.masterPowerOffButtonPushed();

      }
}
```

Aufgabe 3. *Entwurfsmuster* (6 Punkte)

Für die Verwaltung der Termine, die im Rahmen des Bildungsstreiks anstehen, hat die Bonner Bildungsstreikbewegung 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.

a) 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();
	}


}
```

b) 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);
        }
}
```