Übungen zur Vorlesung Softwaretechnologie

- Wintersemester 2017/18 -

Dr. Günter Kniesel

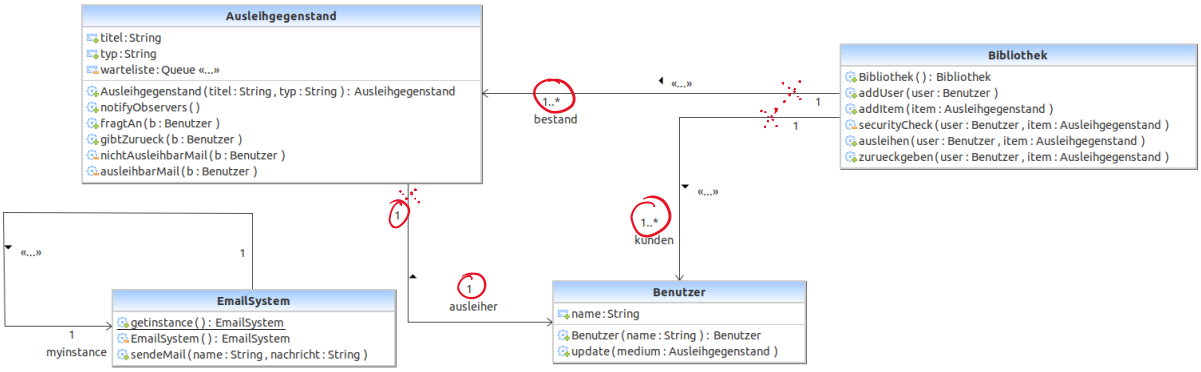
Übungsblatt 9 - Lösungen

Aufgabe 1. *Entwurfsmuster* (20 Punkte)

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

a) *Reengineering (5 Punkte)*:
Erzeugen Sie ein Klassendiagramm des Quellcodes des Bibliotheksverwaltungs-Projekts mit Hilfe von UML-Lab. Exportieren Sie das generierte Modell („File > Export Diagram as Image…“). Machen Sie sich Gedanken, ob das Ergebnis Ihrer Intuition entspricht bzw. ob Sie es anders gemacht hätten. Markieren Sie die „Problemstellen“ im Diagramm (z.B. durch farbige Kringel) und geben Sie das um die Markierungen erweiterte Bild ab, zusammen mit einem Text der beschreibt, was Ihnen aufgefallen ist.

- 2P:



- (1P) 1-en auf der nicht-Pfeil-Seite (z.B. sollte ein Benutzer nicht nur einen Ausleihgegenstand haben dürfen)
- (1P) 1..* bei Liste (könnte auch * sein, sofern die Bibliothek keinen Benutzer oder keinen Ausleihgegenstand hat)
- (1P) 1 wo auch 0..1 Sinn machen würde (Ausleihgenstand hat nicht immer einen Benutzer)
- (1P) Nicht-Navigierbarkeit ist nicht explizit markiert, da das Tool nicht raten kann, ob wir auf konzeptueller Ebene gern in die Gegenrichtung navigieren wollen oder nicht. Daher ist die Navigation an einem Ende offen gelassen (statt explizit ein Kreuz zu machen das die Navigation ausschließt). Beispiel: Assoziation von EmailSystem zu sich selbst.

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 soll 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.

b) *Entwurfsmuster (1 Punkt)*:
Welches Entwurfsmuster bietet sich an, um beliebige Medien-Objekte mit dem statischen Typ *Ausleihgegenstand* erzeugen zu können, ohne dass der konkrete Typ des jeweiligen Objektes schon bei der Entwicklung der Bibliothek bekannt ist?

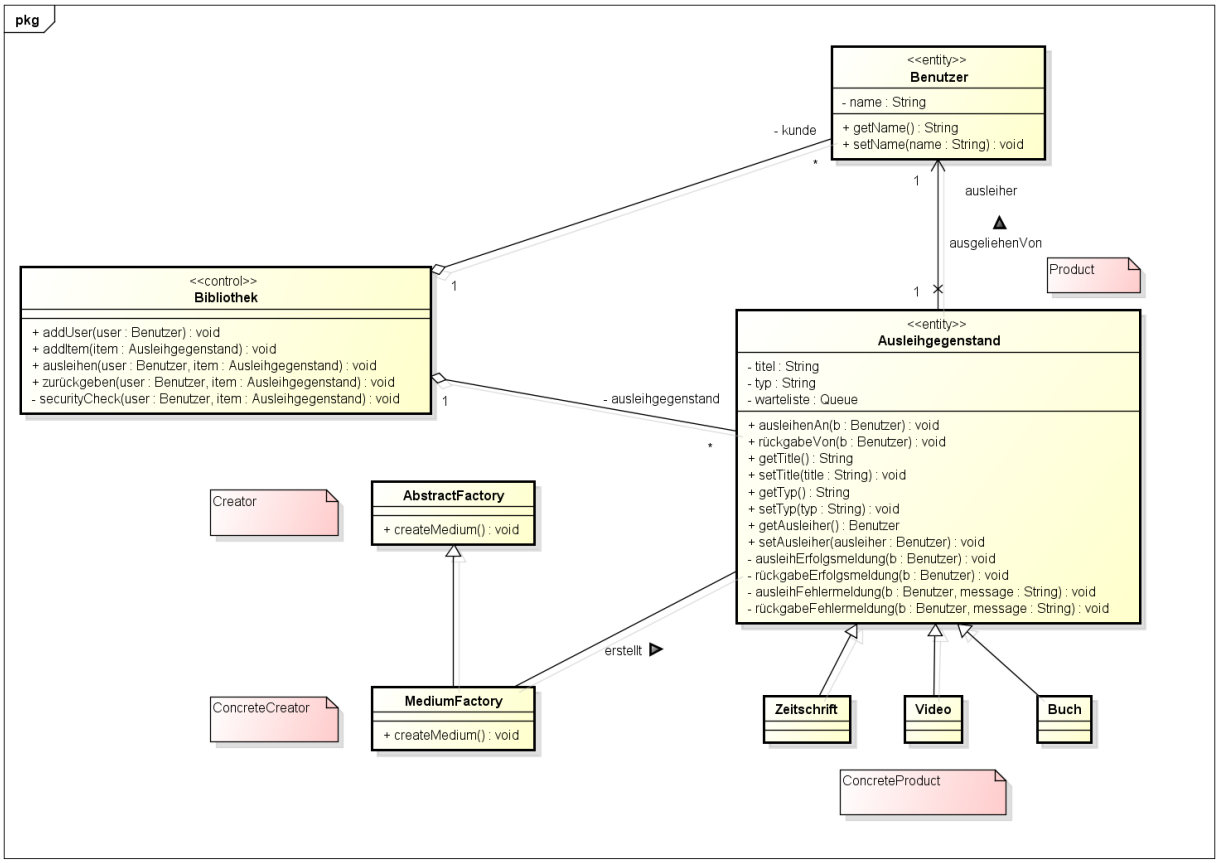
Factory Method

c) *Rollen (2 Punkte)*:
Welche Rollen gibt es in diesem Entwurfsmuster? Wie können Sie diese Rolle bestehenden oder noch zu entwickelnden Klassen des Projektes zuordnen?

Creator (Abstrakt) → neue Klasse „AbstractFactory“
Product (Abstrakt) → vorhandene Klasse „Ausleihgegenstand“
ConcreteCreator → neue Klasse „MediumFactory
ConcreteProduct → neue Klassen „Video“, „Buch“, „Zeitschrift“, etc.

d) *Anwendung des Entwurfsmusters (3 Punkte)*:
Erweitern Sie das Klassendiagramm des Bibliotheksverwaltungs-Projektes um Typen und Methoden die Sie zur Anwendung des Entwurfsmusters zusätzlich brauchen. Entwerfen Sie dabei Klassen für *Video*, *Buch* und *Zeitschrift*, die von der Klasse *Ausleihgegenstand* erben.

e) *Rollenzuordnung (2 Punkte)*:
Markieren Sie in dem erweiterten Diagramm alle Typen und Methoden die eine Rolle in dem Entwurfsmuster spielen mit dem entsprechenden Rollennamen.



f) *Implementierung des Entwurfsmusters (4 Punkte)*:
Implementieren Sie das Entwurfsmuster im Projekt (ohne Änderungen an *Ausleihgegenstand* oder *Bibliothek*).

```
public class Buch extends Ausleihgegenstand {

	public Buch(String titel) {
		super(titel, "Buch");
	}
}

public class Video extends Ausleihgegenstand {

	public Video(String titel) {
		super(titel, "Video");
	}
}

public class Zeitschrift extends Ausleihgegenstand {

	public Zeitschrift(String titel) {
		super(titel, "Zeitschrift");
	}
}

public interface AbstractFactory {
	public Ausleihgegenstand createMedium(String title, String typ);
}
```

```
public class MediumFactory implements AbstractFactory {

	@Override
	public Ausleihgegenstand createMedium(String title, String typ) {
		switch (typ) {
			case "Buch":
				return new Buch(title);
			case "Zeitschrift":
				return new Zeitschrift(title);
			case "Video":
				return new Video(title);
			default:
				return new Ausleihgegenstand(title, typ);
		}
	}
}
```

BibliotheksTest:

```
AbstractFactory fac = new MediumFactory();
item1 = fac.createMedium("Leben des Brian", "Video");
item2 = fac.createMedium("Herr der Ringe", "Video");
item3 = fac.createMedium("How to write unmaintainable Code", "Buch");
item4 = fac.createMedium("Per Anhalter durch die Galaxis", "Buch");
item5 = fac.createMedium("Schöner Wohnen Mai 2006", "Zeitschrift");
```

g) *Nutzung des Entwurfsmusters (4 Punkte)*:
Üben Sie nun, wie Sie auf Basis eines statischen vorgegeben Projektes/Frameworks ein eigenes Projekt aufbauen können. Hier wird nun angenommen, dass die Bibliothek vorgegeben ist (der Quellcode ist also nicht vorhanden, es gibt aber eine entsprechende Dokumentation der Schnittstellen und des Design Patterns). Bauen sie darauf die Uni-Bibliothek auf. Dank des in der Bibliothek verwendeten Patterns geht das inkrementell ohne den Code in der Bibliothek zu verändern. Die Uni-Bibliothek kennt zusätzlich die Typen Seminararbeit und Uralter Wälzer.

```
public class Seminararbeit extends Ausleihgegenstand {

	public Seminararbeit(String titel) {
		super(titel, "Seminararbeit");
	}

}

public class UralterWälzer extends Ausleihgegenstand {

	public UralterWälzer(String titel) {
		super(titel, "Uralter Wälzer");
	}

}

public class UniMediumFactory extends MediumFactory {

	@Override
	public Ausleihgegenstand createMedium(String title, String typ) {
		switch (typ) {
			case "Seminararbeit":
```

```
                        return new Seminararbeit(title);
                    case "Uralter Wälzer":
                        return new UralterWälzer(title);
                    default:
                        break;
                }
                return super.createMedium(title, typ);
        }
}
```

Bibliothekstest:

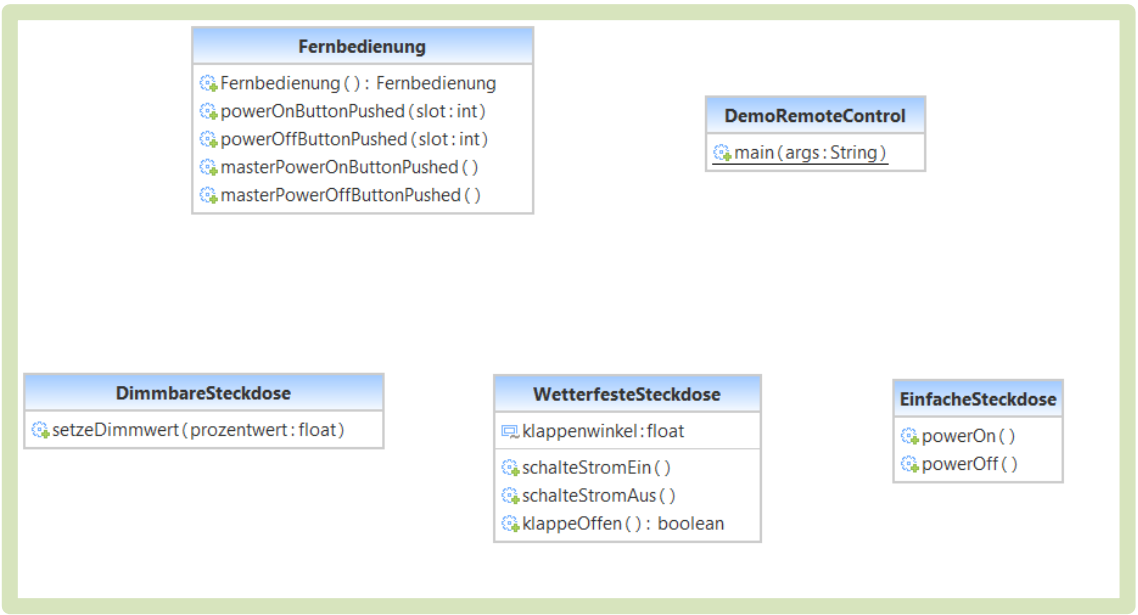
```
AbstractFactory fac = new UniMediumFactory();
item1 = fac.createMedium("Leben des Brian", "Video");
item2 = fac.createMedium("Herr der Ringe", "Video");
item3 = fac.createMedium("How to write unmaintainable Code", "Buch");
item4 = fac.createMedium("Per Anhalter durch die Galaxis", "Buch");
item5 = fac.createMedium("Schöner Wohnen Mai 2006", "Zeitschrift");
item6 = fac.createMedium("Entwicklung eines UML Tools", "Seminararbeit");
item7 = fac.createMedium("Bibel", "Uralter Wälzer");
```

Aufgabe 2. *Entwurfsmuster* (30 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 *readonly*-Repository als *Fernbedienung.zip*.

a) *Klassendiagramm (4 Punkte)*:
Erzeugen Sie ein Klassendiagramm des Projekts. Sie dürfen es manuell oder mit einem Software-Werkzeug machen.



b) *Entwurfsmuster mit Begründung (3 Punkte)*:
Mit welchem Entwurfsmuster ließen sich die folgenden 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.

Begründen Sie Ihre Antwort.

Unten ausgeführt: Command-Pattern (Alternativ: Strategy. Definitiv kein State-Pattern, das ja zentral die Modellierung von Zustandsübergängen beinhaltet.)

c) *Anwendung des Entwurfsmusters (4 Punkte)*:
Erweitern Sie das Klassendiagramm um die benötigten Klassen, um das Entwurfsmuster umzusetzen. Markieren Sie die Rollen der Klassen des Entwurfsmusters im Diagramm.

siehe Lösung von Teilaufgabe f)

d) *Implementierung des Entwurfsmusters (8 Punkte)*:
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[NUMBER_OF_BUTTONROWS];
	Command[] offCommands = new Command[NUMBER_OF_BUTTONROWS];

	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;
	}
}
```

e) *Nutzung des Entwurfsmusters (5 Punkte)*:
Schreiben Sie ein 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();

      }
}
```

f) *Kombination von Mustern (3 Punkte):*
Überlegen Sie (erst mal als UML-Diagramm) wie sie den Entwurf aus Aufgabenteil (c) so anpassen können, dass Sie das Abstract Factory Entwurfsmuster bei der Belegung der Knöpfe einsetzen. Markieren sie jede Klasse, die eine Rolle im Abstract Factory Entwurfsmuster spielt mit der jeweiligen Rolle.

siehe Klassendiagramm am Ende des Dokuments

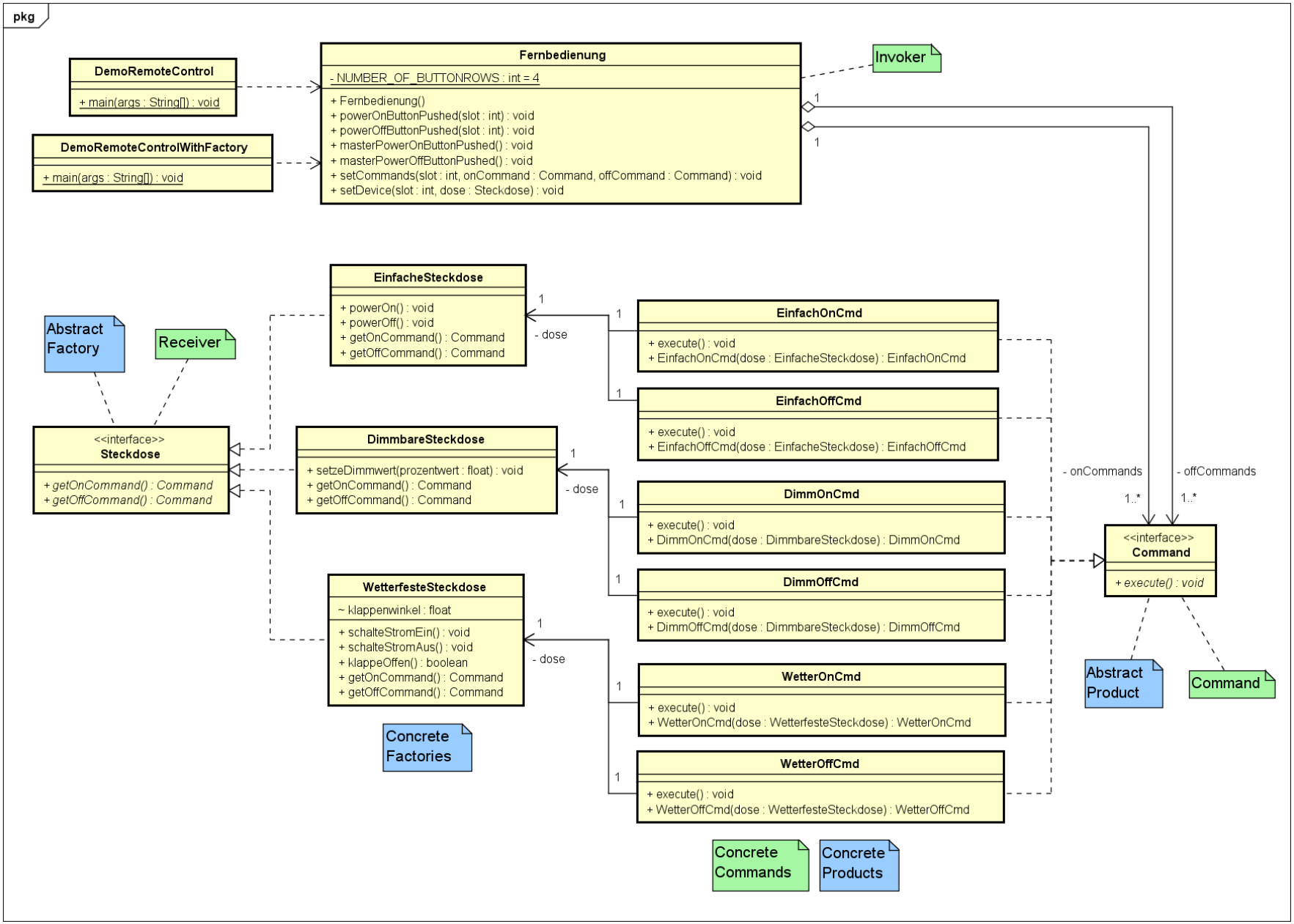
g) *Implementierung des Entwurfsmusters (3 Punkte):*
Kopieren Sie das Programm aus *Aufgabenteil d) + e)* und implementieren Sie dessen laut *Aufgabenteil f)* um die Nutzung des *Abstract Factory* Entwurfsmusters erweiterte Version.

```
public interface Steckdose { //Command-Factory

      Command getOnCommand();
      Command getOffCommand();
}

public class EinfacheSteckdose implements Steckdose {

      […]

      @Override
      public Command getOnCommand() {
            return new EinfacheSteckdoseOnCommand(this);
      }

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

public class DimmbareSteckdose implements Steckdose {

      […]

      @Override
      public Command getOnCommand() {
            return new DimmbareSteckdoseOnCommand(this);
      }

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

```
}

public class WetterfesteSteckdose implements Steckdose {

	[…]

	@Override
	public Command getOnCommand() {
		return new WetterfesteSteckdoseOnCommand(this);
	}

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


public class Fernbedienung {

	[…]

	public void setDevice(int slot, Steckdose dose){
		if ((slot < 0) || (slot >= NUMBER_OF_BUTTONROWS))
			return;

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


public class DemoRemoteControlWithFactory {

	public static void main(String[] args) {

		Fernbedienung fb = new Fernbedienung();

		fb.setDevice(0, new EinfacheSteckdose());
		fb.setDevice(1, new EinfacheSteckdose());
		fb.setDevice(2, new DimmbareSteckdose());
		fb.setDevice(3, new WetterfesteSteckdose());

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

		fb.masterPowerOnButtonPushed();
		fb.masterPowerOffButtonPushed();
	}
}
```

Frage: *Ist der Aufruf der super()-Methode bei der Implementation eines Interfaces (*z.B. in *WetterfesteSteckdoseOnCommand*) *im Konstruktor nicht falsch oder überflüssig?*

Da jede Klasse grundsätzlich von der Klasse *Object* erbt, ist dies nicht falsch. Der Aufruf der super()-Methode im Konstruktor ist sogar empfehlenswert und wird andernfalls durch den Compiler umgesetzt.


pkg
DemoRemoteControl
+ main(args : String[]) : void
DemoRemoteControlWithFactory
+ main(args : String[]) : void
Fernbedienung
- NUMBER_OF_BUTTONROWS : int = 4
+ Fernbedienung()
+ powerOnButtonPushed(slot : int) : void
+ powerOffButtonPushed(slot : int) : void
+ masterPowerOnButtonPushed() : void
+ masterPowerOffButtonPushed() : void
+ setCommands(slot : int, onCommand : Command, offCommand : Command) : void
+ setDevice(slot : int, dose : Steckdose) : void
Invoker
1
1
Abstract Factory
Receiver
<<interface>>
Steckdose
+ getOnCommand() : Command
+ getOffCommand() : Command
EinfacheSteckdose
+ powerOn() : void
+ powerOff() : void
+ getOnCommand() : Command
+ getOffCommand() : Command
DimmbareSteckdose
+ setzeDimmwert(prozentwert : float) : void
+ getOnCommand() : Command
+ getOffCommand() : Command
WetterfesteSteckdose
~ klappenwinkel : float
+ schalteStromEin() : void
+ schalteStromAus() : void
+ klappeOffen() : boolean
+ getOnCommand() : Command
+ getOffCommand() : Command
Concrete Factories
1
- dose
1
1
- dose
1
1
- dose
1
EinfachOnCmd
+ execute() : void
+ EinfachOnCmd(dose : EinfacheSteckdose) : EinfachOnCmd
EinfachOffCmd
+ execute() : void
+ EinfachOffCmd(dose : EinfacheSteckdose) : EinfachOffCmd
DimmOnCmd
+ execute() : void
+ DimmOnCmd(dose : DimmbareSteckdose) : DimmOnCmd
DimmOffCmd
+ execute() : void
+ DimmOffCmd(dose : DimmbareSteckdose) : DimmOffCmd
WetterOnCmd
+ execute() : void
+ WetterOnCmd(dose : WetterfesteSteckdose) : WetterOnCmd
WetterOffCmd
+ execute() : void
+ WetterOffCmd(dose : WetterfesteSteckdose) : WetterOffCmd
Concrete Commands
Concrete Products
- onCommands
1..*
- offCommands
1..*
<<interface>>
Command
+ execute() : void
Abstract Product
Command