

Studiengang: Wirtschaftsinformatik Bachelor



**Westfälische  
Hochschule**

Sommersemester 2017

4. Fachsemester

# **Architektur betrieblicher Informationssysteme**

## **Semesteraufgabe 6 - Analysemuster II**

Abgabedatum:

28. Juni 2017

Aktualisierungsdatum:

18. Juli 2017

Autoren:

Mehmet Tüfekci (Matr. 201521617) – [Mehmet.Tuefekci@studmail.w-hs.de](mailto:Mehmet.Tuefekci@studmail.w-hs.de)

Mario Kellner (Matr. 201520916) – [Mario.Kellner@studmail.w-hs.de](mailto:Mario.Kellner@studmail.w-hs.de)

Julian Kranen (Matr. 201223532) – [Julian.Kranen@studmail.w-hs.de](mailto:Julian.Kranen@studmail.w-hs.de)

## Aufgabe 1 (Analysemuster)

### Teilaufgabe A)

**Die Prinzipien der Muster Quantität und Umwandlungsverhältnis kann auch auf Währungen angewendet werden. Beschreiben Sie wie dies erfolgen kann? Welche Erweiterungen sind eventuell erforderlich?**

Die Geldangabe in einer Währung ist eine Quantität, denn der Betrag in z. B. Euro hat einen Wert in Euro-Einheiten.

Zum Beispiel 13,00 € haben den Wert von 13, der Einheit Euro.

Umwandlungsverhältnisse existieren innerhalb einer Währung, z. B. zwischen Euro und Cent, dabei entsprechen 13,00 € genau 1.300 Cent.

Umwandlungsverhältnisse existieren aber auch zwischen Währungen, z.B. entspricht 1,00 € in etwa 1,12 US-Dollar.

Hierbei muss jedoch eine Zeitangabe hinzugefügt werden, da sich der Umrechnungskurs zwischen Währungen ständig ändert.

### Teilaufgabe B)

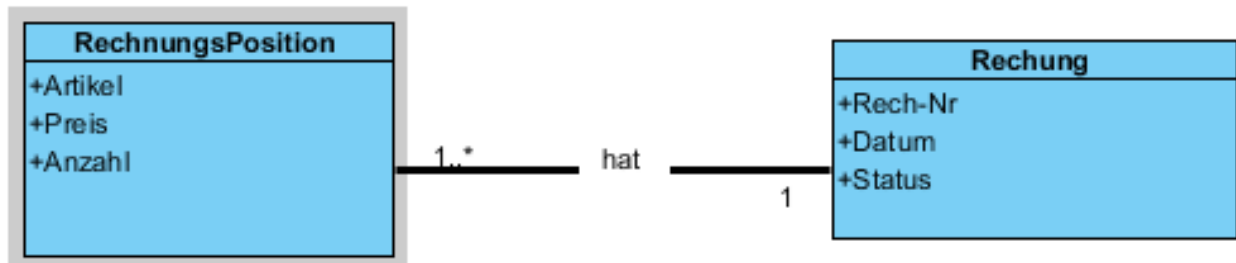
**Was bedeutet die Unterscheidung zwischen Wissensebene und operationeller Ebene beim Muster „Messung“?**

Die beiden Ebenen unterscheiden sich dahingehen, dass die Wissensebene festhält, was gemessen werden kann, z. B. Größe, Gewicht. Die operationale Ebene hält konkrete Messungen, z. B. einer Größe, fest, die gemessen werden kann.

Aufgabe 2 (Unterstützungsmuster)

## Aufgabe 2 (Unterstützungsmuster)

Für die Entwurfsschablone „Assoziation“ soll folgende Situation abgebildet werden: Eine Rechnung hat eine Beziehung zu ein oder mehr Rechnungspositionen. Jede Rechnungsposition gehört zu genau einer Rechnung. Erstellen Sie ein UML-Klassendiagramm.



## Aufgabe 3 (Erzeugungsmuster)

Nehmen Sie an, dass Sie ein Java-Programm erstellen müssen, in dem Sie eine Klasse „Fahrzeuge“ benutzen wollen. Allerdings werden später als Objekte unterschiedliche Arten von Fahrzeugen, nämlich „Fahrräder“ und „Autos“ vorkommen, die auch zum Teil voneinander abweichende Attribute und unterschiedliche Realisierungen derselben Methoden haben werden. Es ist auch damit zu rechnen, dass später weitere Fahrzeugklassen (z. B. Tretroller) hinzukommen. Erstellen Sie eine einfache Implementierung, in der die Muster „Abstrakte Fabrik“ und „Singleton“ verwendet werden (dabei soll „Singleton“ nicht im Zusammenhang mit der Klasse „Fahrzeuge“ verwendet werden).

Abzugeben sind der Quelltext und ein Beispielablauf, der ein Fahrrad und ein Auto zeigt.

```
1 package Main;
2 import Entities.Fahrzeug;
3 import Factory.AbstractFactory;
4 import Factory.FactoryCreator;
5
6 public class Main {
7     public static void main(String[] args) {
8         // Zum verdeutlichen Variablen explizit deklariert
9         AbstractFactory factory;
10        Fahrzeug fahrzeug;
11
12        factory = FactoryCreator.getInstance().createfactory("Auto");
13        fahrzeug = factory.create(new String[]{"Silber", "VW Passat", "106"});
14
15        fahrzeug.beschreibung();
16        fahrzeug.beschleunigen();
17        fahrzeug.hupe();
18        fahrzeug.bremsen();
19
20        factory = FactoryCreator.getInstance().createfactory("Fahrrad");
21        fahrzeug = factory.create(new String[]{"Grün", "27", "Scheibenbremsen"});
22
23        fahrzeug.beschreibung();
24        fahrzeug.beschleunigen();
25        fahrzeug.hupe();
26        fahrzeug.bremsen();
27
28    }
29 }
```

Listing 1: Main.java

```
1 package Entities;
2
3 public interface Fahrzeug {
4     public void beschreibung();
```

Aufgabe 3 (Erzeugungsmuster)

```
5 public void bremsen();  
6 public void beschleunigen();  
7 public void hupe();  
8 }
```

Listing 2: Fahrzeug.java

```
1 package Entities;  
2  
3 public class Fahrrad implements Fahrzeug {  
4     private String Farbe;  
5     private String anzahlDerGaenge;  
6     private String BremsenArt;  
7  
8     public Fahrrad(String farbe, String anzahlDerGaenge, String bremsenArt) {  
9         setFarbe(farbe);  
10        setAnzahlDerGaenge(anzahlDerGaenge);  
11        setBremsenArt(bremsenArt);  
12    }  
13  
14    public String getFarbe() {  
15        return Farbe;  
16    }  
17  
18    public void setFarbe(String farbe) {  
19        Farbe = farbe;  
20    }  
21  
22    public String getAnzahlDerGaenge() {  
23        return anzahlDerGaenge;  
24    }  
25  
26    public void setAnzahlDerGaenge(String anzahlDerGaenge) {  
27        this.anzahlDerGaenge = anzahlDerGaenge;  
28    }  
29  
30    public String getBremsenArt() {  
31        return BremsenArt;  
32    }  
33  
34    public void setBremsenArt(String bremsenArt) {  
35        BremsenArt = bremsenArt;  
36    }  
37  
38    @Override  
39    public void bremsen() {  
40        System.out.println("[Fahrrad] Der Radfahrer bremst!");  
41    }  
42  
43    @Override  
44    public void beschleunigen() {
```

Aufgabe 3 (Erzeugungsmuster)

```
45     System.out.println("[Fahrrad] Beschleunigen. Immer Flotter und immer weiter, so der Radfahrer  
         heiter!");  
46 }  
47  
48 @Override  
49 public void beschreibung() {  
50     System.out.println("[Fahrrad] Farbe = " + getFarbe() + ", BremsenArt = " + getBremsenArt() + ",  
         Anzahl der Gänge = " + getAnzahlDerGaenge());  
51 }  
52  
53 @Override  
54 public void hupe() {  
55     System.out.println("[Fahrrad] Ringring");  
56 }  
57 }
```

Listing 3: Fahrrad.java

```
1 package Entities;  
2  
3 public class Auto implements Fahrzeug {  
4     private String Farbe;  
5     private String Marke;  
6     private int Leistung;  
7  
8     public Auto(String Farbe, String Marke, int Leistung) {  
9         setFarbe(Farbe);  
10        setMarke(Marke);  
11        setLeistung(Leistung);  
12    }  
13  
14    public String getFarbe() {  
15        return Farbe;  
16    }  
17  
18    public void setFarbe(String farbe) {  
19        Farbe = farbe;  
20    }  
21  
22    public String getMarke() {  
23        return Marke;  
24    }  
25  
26    public void setMarke(String marke) {  
27        Marke = marke;  
28    }  
29  
30    public int getLeistung() {  
31        return Leistung;  
32    }  
33 }
```

Aufgabe 3 (Erzeugungsmuster)

```
34 public void setLeistung(int leistung) {
35     Leistung = leistung;
36 }
37
38 @Override
39 public void bremsen() {
40     System.out.println("[Auto] Die Reifen quitschen laut während des Bremsens.");
41 }
42
43 @Override
44 public void beschleunigen() {
45     System.out.println("[Auto] Der Fahrer gibt Vollgas");
46 }
47
48 @Override
49 public void beschreibung() {
50     System.out.println("[Auto] Farbe = " + getFarbe() + ", Marke = " + getMarke() + ", Leistung = "
51         + getLeistung());
52 }
53
54 @Override
55 public void hupe() {
56     System.out.println("[Auto] Moooooop");
57 }
58 }
59 }
```

Listing 4: Auto.java

```
1 package Factory;
2
3 import Entities.Fahrzeug;
4
5 public abstract class AbstractFactory {
6     public abstract Fahrzeug create(String[] FactoryParameters);
7 }
```

Listing 5: AbstractFactory.java

```
1 package Factory;
2
3 import Entities.Fahrrad;
4 import Entities.Fahrzeug;
5
6 public class FahrradFactory extends AbstractFactory {
7     @Override
8     public Fahrzeug create(String[] FactoryParameters) {
9         if(FactoryParameters.length != 3) return null;
10
11         return new Fahrrad(FactoryParameters[0], FactoryParameters[1], FactoryParameters[2]);
12     }
13 }
```

## Aufgabe 3 (Erzeugungsmuster)

```
12 }  
13 }
```

Listing 6: FahrradFactory.java

```
1 package Factory;  
2  
3 import Entities.Auto;  
4 import Entities.Fahrzeug;  
5  
6 public class AutoFactory extends AbstractFactory {  
7     @Override  
8     public Fahrzeug create(String[] FactoryParameters) {  
9         if(FactoryParameters.length != 3) return null;  
10  
11         return new Auto(FactoryParameters[0], FactoryParameters[1], Integer.parseInt(FactoryParameters  
12             [2]));  
13     }  
14 }
```

Listing 7: AutoFactory.java

```
1 package Factory;  
2  
3 public class FactoryCreator {  
4     // Singleton Pattern  
5     public static FactoryCreator instance;  
6  
7     private FactoryCreator() {  
8         // Leerer privater Konstruktor  
9     };  
10  
11     public static FactoryCreator getInstance() {  
12         if(instance == null) {  
13             instance = new FactoryCreator();  
14         }  
15  
16         return instance;  
17     }  
18  
19     public AbstractFactory createfactory(String FactoryType) {  
20         if(FactoryType.equals("Fahrrad")) {  
21             return new FahrradFactory();  
22         }  
23  
24         if(FactoryType.equals("Auto")) {  
25             return new AutoFactory();  
26         }  
27  
28         return null;  
29     }  
30 }
```

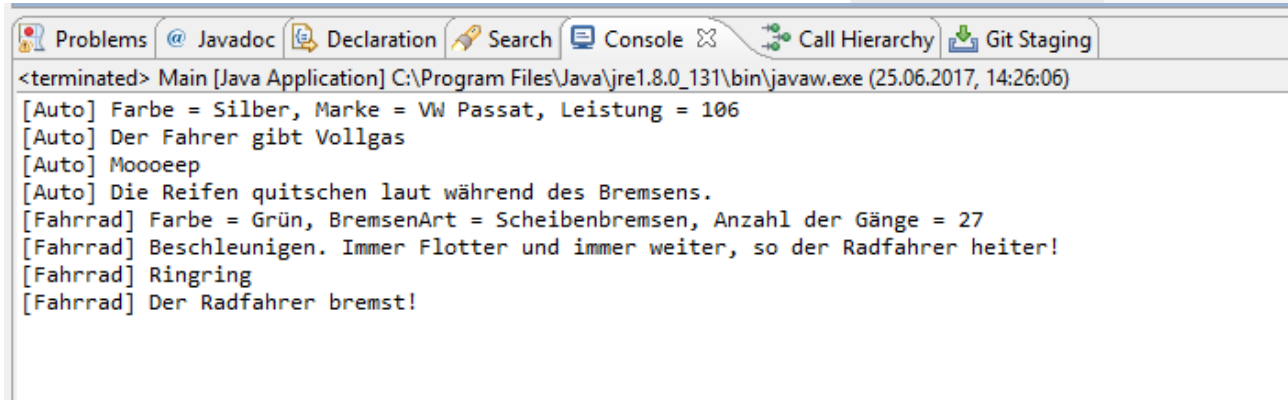


Aufgabe 3 (Erzeugungsmuster)

30 }

Listing 8: FactoryCreator.java

Ausgabe der Konsole:



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (25.06.2017, 14:26:06)
[Auto] Farbe = Silber, Marke = VW Passat, Leistung = 106
[Auto] Der Fahrer gibt Vollgas
[Auto] Moooeep
[Auto] Die Reifen quitschen laut während des Bremsens.
[Fahrrad] Farbe = Grün, BremsenArt = Scheibenbremsen, Anzahl der Gänge = 27
[Fahrrad] Beschleunigen. Immer Flotter und immer weiter, so der Radfahrer heiter!
[Fahrrad] Ringring
[Fahrrad] Der Radfahrer bremst!
```