

# Klassenenentwurf I

## Lernziele

- Sie können Programme im Umfang von einigen Klassen bezüglich Kohäsion, Codeduplizierung, Kopplung und Kapselung analysieren und verbessern.

## Aufgabe 1 (auf Papier!)

Das nachfolgende Codebeispiel weist eine geringe Kohäsion auf:

```
public class Kohaesion {  
  
    private int zahl = 1;  
  
    public int manipuliereZahl(int befehl, int zahl) {  
        if (befehl == 0) {  
            this.zahl = zahl;  
        }  
        if (befehl == 1) {  
            this.zahl = this.zahl + zahl;  
        }  
        if (befehl == 2) {  
            this.zahl = this.zahl - zahl;  
        }  
        return this.zahl;  
    }  
}
```

Woran erkennen Sie dies und wieso ist das schlecht?

Die Methode `manipuliereZahl` führt je nach Eingabe eine oder keine von drei Operationen aus. Sie hat also eine schwache Kohäsion. Im Allgemeinen ist das schlecht, weil so Übersichtlichkeit und Wiederverwendbarkeit von Klassen und Methoden verloren geht.

In diesem Beispiel muss man die internen Details der Methode kennen (die richtige Zahl, um die gewünschte Operation auszuführen).

Machen Sie es besser: Wie müsste die Klasse aussehen, damit sie eine hohe Kohäsion aufweist?  
Sie dürfen die Klasse inkl. Schnittstelle komplett neu schreiben.

Siehe separates Bild aufgabe1.png

## Aufgabe 2 (auf Papier!)

Wir betrachten eine Applikation, welche eine Eingabemaske für die Neueingabe und eine weitere Eingabemaske für die Änderung von Benutzerdaten verwendet. Auf der folgenden Seite finden Sie zwei Vorschläge für Programmcode, der für die Entgegennahme und Weiterleitung der gemachten Benutzereingaben an einen sog. Controller zuständig ist. Welche Lösung würden Sie vorziehen? Mit welcher Begründung?

Variante 1 hat eine Klasse, die sowohl Änderungen als auch Neueingaben behandelt, in Variante 2 wurden dafür zwei separate Klassen erstellt. Somit ist die Kohäsion in Variante 2 stärker als in Variante 1, da die Anforderungen auf zwei Klassen verteilt wurden. Variante 2 ist deshalb vorzuziehen.

Variante 1:

```
public class Ansicht {  
  
    // Nicht gezeigte Methoden und Datenfelder der Ansicht  
  
    public void nameEingeben(String name) {  
        kontrollier.nameEingeben(name);  
    }  
  
    public void adresseEingeben(String adresse) {  
        kontrollier.adresseEingeben(adresse);  
    }  
  
    public void nameAendern(String nameAlt, String nameNeu) {  
        kontrollier.nameAendern(nameAlt, nameNeu);  
    }  
  
}
```

Variante 2:

```
public class AnsichtEingeben {  
  
    // Nicht gezeigte Methoden und Datenfelder der Ansicht  
  
    public void nameEingeben(String name) {  
        kontrollier.nameEingeben(name);  
    }  
  
    public void adresseEingeben(String adresse) {  
        kontrollier.adresseEingeben(adresse);  
    }  
  
}  
  
public class AnsichtAendern {  
  
    // Nicht gezeigte Methoden und Datenfelder der Ansicht  
  
    public void nameAendern(String nameAlt, String nameNeu) {  
        kontrollier.nameAendern(nameAlt, nameNeu);  
    }  
  
}
```

## Aufgabe 3

Importieren Sie für diese Aufgabe das BlueJ-Projekt 06\_Praktikum-1\_Zuul-schlecht und machen Sie sich mit dem Programm und dem Quellcode vertraut.

In der `Spiel` Klasse gibt es doppelten Code. Finden Sie diesen Code und schreiben Sie eine neue Methode, so dass der Code nicht mehr redundant vorhanden ist. Die Methode sollte folgende Signatur haben:

```
private void rauminfoAusgeben()
```

Hinweis: Beim gesuchten Code handelt es sich um Systemausgaben die in den Methoden `wechsleRaum` und `willkommenstextAusgeben` zu finden sind.

## Aufgaben 4

Untersuchen Sie sich nun die Klasse `Raum`. Darin finden sich folgende Zeilen:

```
public String beschreibung;  
public Raum nordausgang;  
public Raum suedausgang;  
public Raum ostausgang;  
public Raum westausgang;
```

Bezüglich der Kopplung sowie der Kapselung ist das ein schlechter Klassenentwurf. Erklären Sie wieso dem so ist:

Benutzen Sie für die Räume nun statt der obigen Variante eine `HashMap` und setzen alle Datenfelder der Klasse auf `private`.

```
private String beschreibung;  
private HashMap<String, Raum> ausgaenge;
```

Passen Sie anschliessend die ganze `Raum` Klasse entsprechend an. Unter anderem sollte die angepasste Klasse die folgenden beiden Methodensignaturen aufweisen:

```
public void setzeAusgaenge(String richtung, Raum raum)  
  
public Raum gibAusgang(String richtung)
```

Passen Sie schliesslich noch die Klasse `Spiel` an die neue Situation an.

## Aufgabe 5

In der Aufgabe 3 haben Sie die Methode `rauminfoAusgeben` in der Klasse `Spiel` erstellt. Wenn Sie sich diese Methode genauer ansehen, bereitet diese Klasse Informationen auf, die von der Klasse `Raum` verwaltet werden.

Ein besseres Design ist, wenn diese Informationen in der Klasse `Raum` aufbereitet werden und von der Klasse `Spiel` abgerufen werden. Schreiben Sie eine solche Methode in der Klasse `Raum` mit der folgenden Signatur.

```
/**
 * Liefere eine Beschreibung der Ausgänge dieses Raumes, beispielsweise
 * "Ausgänge: north west".
 *
 * @return eine Beschreibung der verfügbaren Ausgänge
 */
public String gibAusgaengeAlsString()
```

Erklären Sie, wieso dieses Design besser ist:

## Aufgabe 6

In der letzten Aufgabe haben Sie die Methode `gibAusgaengeAlsString` in der Klasse `Raum` erstellt. Bezüglich Ausgabe verbleibt trotz dieser Änderung noch eine Kopplung, die sich entfernen lässt. Finden und entfernen Sie diese Kopplung. Dazu müssen Sie unter anderem die nachfolgende Methode zur Klasse `Raum` hinzufügen:

```
/**
 * Liefere eine lange Beschreibung dieses Raumes, in der Form
 *
 * Sie sind in der Küche
 * Ausgänge: north west
 *
 * @return eine lange Beschreibung dieses Raumes
 */
public String gibLangeBeschreibung()
```

## Aufgabe 7

Zeichnen Sie ein Objektdiagramm, dass alle Objekte Ihres Spiels zu dem Zeitpunkt zeigt, zu dem das Spiel gerade gestartet wurde.

