Software Engineering 2
Übungsblatt 7
– Refactoring –

Ausgabe: 17.11.2022 **Besprechung: 24.11.2022 und 28.11.2022**

Ablauf der Übungen

- Erfolgreiches Bearbeiten von 2/3 der Pflichtübungsaufgaben und persönliche Vorstellung mindestens einer Aufgabenlösung (auf Nachfrage) berechtigen zur Klausurteilnahme.
- Eine Aufgabe gilt als erfolgreich bearbeitet, wenn eine korrekte Lösung bzw. ein nachvollziehbarer Lösungsversuch termingerecht abgegeben wurden.
- Die Aufgaben sollen in 2er Gruppen bearbeitet werden.
- Die Abgabe erfolgt per Moodle als PDF-Datei vor der Besprechung in der Übung.
- Das PDF-Dokument enthält Namen und Matrikelnummern der an der Lösung beteiligten Personen.
- Das (identische) PDF-Dokument wird von allen an der Lösung beteiligten Personen auf Moodle hochgeladen.
- Das Ergebnis soll in der Übung vorgestellt und diskutiert werden.

Aufgabe 7.1: Refactoring – Entkoppeln von Klassen

Gegeben ist folgendes Java-Programmfragment:

```
class Servicenutzer {
    Serviceanbieter s;
    ...
    void meth() {
        s.a();
    }
}

class Serviceanbieter {
    public void a(){...}
    public void b(){...}
}
```

1. Beschreiben Sie, welche Abhängigkeiten zwischen den Klassen `Servicenutzer` und `Serviceanbieter` bestehen?
2. Entkoppeln Sie die Klassen durch Verwendung der Refactoring-Methode "**Extract Interface**" (siehe Kap. 6.2).

Aufgabe 7.2: Refactoring – Fallunterscheidung durch Polymorphie ersetzen (Pflichtaufgabe)

Gegeben ist folgendes Java-Programmfragment, in dem die Berechnung eines Kaufpreises für ein Kleidungsstück vom Typ (`NORMAL, SALE, SUPERSALE`) des Verkaufs abhängt.

```
class Garment {
    ...
    double getPrice() {
        switch (_type) {
            case NORMAL:
                return getNormalPrice();
            case SALE:
                return getNormalPrice() * getDiscount();
            case SUPERSALE:
                return getNormalPrice() * 0.5;
        }
    }
}
```

Wenden Sie die Refactoring-Methode "**Replace Conditional with Polymorphism**" (siehe Kap. 6.2) auf das obige Beispiel an. Zeichnen Sie das Klassendiagramm vor und nach der Anwendung der Refactoring-Methode. Skizzieren Sie die Implementierung nach dem Refactoring-Schritt.

Aufgabe 7.3: Refactoring – Bad Smell: Temporäre Nutzung von Attributen (Temporary Fields)
An einer Hochschule müssen für jede*n Student*in neben den persönlichen Daten auch die Bankverbindung gespeichert werden, damit die Semesterbeiträge per Lastschrift abgebucht werden können:

```
public class Student {

    private String vorname, nachname;
    private String geldinstitut;
    private String iban;
    ...
}
```

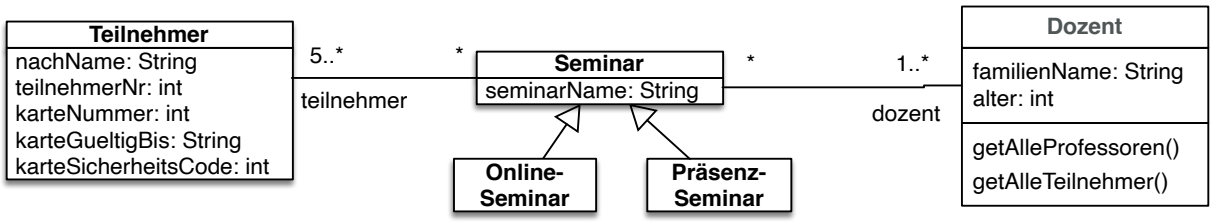
Als neue Anforderung muss es möglich sein, dass sich die Bankverbindung ändert kann: Bis zu einem bestimmten Termin soll dann noch die bisherige Bankverbindung gelten, aber nach dem Änderungstermin sollen die Beiträge von einem neuen Konto abgebucht werden. Die Klasse wird dementsprechend erweitert:

```
import java.util.*;

public class Student2 {

    private String vorname, nachname;
    private String geldinstitut;
    private String iban;

    private String geldinstitut2;
    private String iban2;
    private Date gueltigAb; // fuer Konto2
    ...
}
```

Welche Nachteile hat diese neue zweite Lösung? Wie lässt sich die Struktur verbessern, ohne dass sich die Funktionalität verändert?

Aufgabe 7.4: Refactoring

Welche Refactoring-Methoden sollten an dem nachfolgenden Klassenmodell durchgeführt werden?



(karte = Kreditkarte)

Aufgabe 7.5: Refactoring (Pflichtaufgabe)

Die nachfolgende Klasse stellt eine Linie im zweidimensionalen Raum dar (Source-Datei steht auf Moodle).

```
4  public class Linie {
5
6      private double startX;
7      private double startY;
8
9      private double endX;
10     private double endY;
11
12     public Linie(double startX, double startY,
13             double endX, double endY) {
14         this.startX = startX;
15         this.startY = startY;
16         this.endX = endX;
17         this.endY = endY;
18     }
19
20     public double distanz() {
21         return Math.sqrt(Math.pow(endX - startX, 2.0) + Math.pow(endY - startY, 2.0));
22     }
23
24     public double flaeche() {
25         return (Math.abs(endX - startX) * Math.abs(endY - startY)) / 2;
26     }
27 }
```

1. Schreiben Sie eine einfache Testklasse (mit main-Methode), mit der Sie das Verhalten der Klasse Linie überprüfen können.
2. Führen Sie mindestens zwei unterschiedliche Refactoring-Methoden auf der Klasse aus. Arbeiten Sie hierzu auch die Refactoring-Methoden aus dem Skript durch, die nicht explizit in der Vorlesung besprochen wurden. Bitte beachten Sie, dass in jedem Schritt nur **eine** Änderung erfolgen darf und danach alle Tests erfolgreich durchlaufen werden müssen.