Abgabe SE2 Übung 7

Furkan Hidayet Rafet Aydin, M1630039
Falk-Niklas Heinrich, M1630123

2022-11-19

Aufgabe 7.1: Refactoring – Entkoppeln von Klassen

```
1  class Serviceanbieter {
2     public void a() {
3        // ...
4     }
5
6     public void b() {
7        // ...
8     }
9  }
10
11 class Servicenutzer {
12    Serviceanbieter s;
13
14    // ...
15
16    public void meth() {
17       s.a();
18    }
19 }
```

1. Abhängigkeiten zwischen den Klassen `Servicenutzer` und `Serviceanbieter`

Die Klasse `Servicenutzer` kennt und hängt direkt ab von `Serviceanbieter`. Es besteht daher eine enge Kopplung, außerdem nutzt `Servicenutzer` nur einen kleinen Teil von `Serviceanbieter`.

2. Entkoppeln Sie die Klassen durch Verwendung der Refactoring-Methode "Extract Interface"

```
1  interface MethA {
2     void a();
3  }
4
5  class Serviceanbieter implements MethA {
6     public void a() {
7        // ...
8     }
9
10    public void b() {
11       // ...
12    }
13 }
14
15 class Servicenutzer {
16    MethA s;
17
18    public void meth() {
19       s.a();
20    }
21 }
```

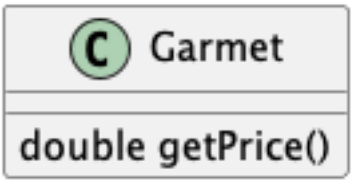

Abbildung 1: UML Klassendiagramm vor Refactoring

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

UML vor Refactoring

UML nach Refactoring

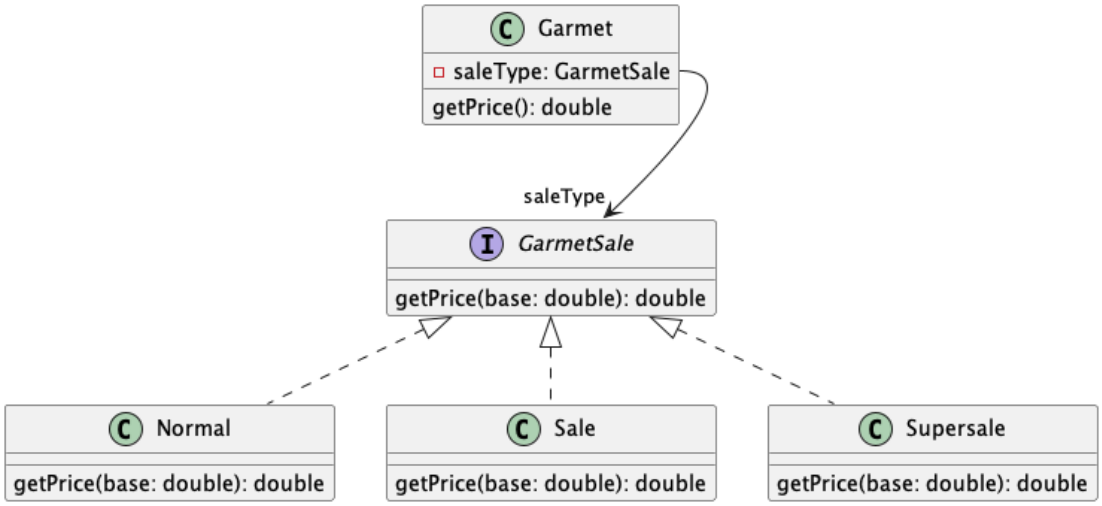

Abbildung 2: UML Klassendiagramm nach Refactoring

Implementierung (Skizziert)

```
1  interface GarmetSale {
2      double getPrice(double base);
3  }
4
```

```
5  class Normal implements GarmetSale {
6     double getPrice(double base) {
7        return base;
8     }
9  }
10
11 class Sale implements GarmetSale {
12    double getPrice(double base) {
13       return base * 0.85;
14    }
15 }
16
17 class Supersale implements GarmetSale {
18    double getPrice(double base) {
19       return base * 0.75;
20    }
21 }
22
23 class Garmet {
24    GarmetSale saleType;
25
26    double getPrice() {
27       return this.saleType.getPrice(this.basePrice);
28    }
29 }
```

Aufgabe 7.3: Refactoring – Bad Smell: Temporäre Nutzung von Attributen (Temporary Fields)

Nachteile der zweiten Lösung

- Schlecht Erweiterbar (nochmalige Änderung der Bankverbindung?).
- Keine klare Zuständigkeit für Instanzvariablen → führt schnell zu inkonsistentem Zustand.
- Verletzt das Prinzip des Information Hiding.
- Schwache Kohärenz: Nicht Kernaufgabe der Klasse

Verbesserungsvorschlag

Extract Class anwenden und die Bankverbindungen in eine eigene Klasse auslagern. Diese Klasse kann dann die Bankverbindung zurückgeben, welche aktuell korrekt ist.

Aufgabe 7.4: Refactoring

- **Extract Class**

  Karteninformationen aus `Teilnehmer` auslagern.

- **Move Method**

  `getAlle*()`-Methoden von `Dozent` nach `Seminar` verschieben.

- **Replace Subclass with Delegate**

  `OnlineSeminar` und `PräsenzSeminar` sollten nicht als Unterklassen ausgeführt sein. Durch die Vererbung können die beiden verbundenen Klassen `Teilnehmer` und `Dozent` nur kompliziert und mit zusätzlichem Wissen auf die Subklassenfunktionen zugreifen. Beispiel: `Dozent`-Objekt möchte wissen ob eines seiner Seminare online oder in Präsenz abzuhalten ist. `Dozent` benötigt zusätzlichen Wissen über Unterklassen von `Seminar`, um dann mittels `typeof` oder Ähnlichem den Typ zu bestimmen.

- **Rename Field**

  `Teilnehmer::nachName` vs. `Dozent::familienName`.

Aufgabe 7.5: Refactoring (Pflichtaufgabe)

Klasse `Linie`

```
1  public class Linie {
2      private double startX;
3      private double startY;
4
5      private double endX;
6      private double endY;
```

```
7
8       public Linie(double startX, double startY, double endX, double endY) {
9           this.startX = startX;
10          this.startY = startY;
11          this.endX = endX;
12          this.endY = endY;
13      }
14
15      public double distanz() {
16          return Math.sqrt(Math.pow(endX - startX, 2.0) + Math.pow(endY - startY,
            ↪  2.0));
17      }
18
19      public double flaeche() {
20          return (Math.abs(endX - startX) * Math.abs(endY - startY)) / 2.0;
21      }
22  }
```

Test

```
1   public class LinieTest {
2
3       public static void main(String[] args) {
4           testConstructor();
5           testDistanz();
6           testFlaeche();
7       }
8
9       private static void testConstructor() {
10          Linie[] cases = {
11                  new Linie(0.0, 0.0, 0.0, 0.0),
12                  new Linie(-1.0, -1.0, -1.0, -1.0),
13                  new Linie(Double.MAX_VALUE, Double.MIN_VALUE,
                    ↪  Double.MIN_NORMAL, -Double.MAX_VALUE),
14                  new Linie(new Linie.Point(0.0, 0.0), new Linie.Point(0.0,
                    ↪  0.0)),
15                  new Linie(new Linie.Point(-1.0, -1.0), new Linie.Point(-1.0,
                    ↪  -1.0)),
16                  new Linie(new Linie.Point(Double.MAX_VALUE, Double.MIN_VALUE),
```

```
                        new Linie.Point(Double.MIN_NORMAL, -Double.MAX_VALUE))
        };
        for (Linie linie : cases) {
            assert linie != null;
        }
    }

    private static void testDistanz() {
        assert Double.compare((new Linie(0.0, 0.0, 0.0, 0.0)).distanz(), 0.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, 2.0, 0.0)).distanz(), 2.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, -2.0, 0.0)).distanz(), 2.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, -2.0, -2.0)).distanz(),
        ↪  Math.sqrt(8.0)) == 0;
        assert Double.compare((new Linie(-2.0, -2.0, -2.0, -2.0)).distanz(),
        ↪  0.0) == 0;
    }

    private static void testFlaeche() {
        assert Double.compare((new Linie(0.0, 0.0, 0.0, 0.0)).flaeche(), 0.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, 2.0, 0.0)).flaeche(), 0.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, -2.0, 0.0)).flaeche(), 0.0)
        ↪  == 0;
        assert Double.compare((new Linie(0.0, 0.0, -2.0, -2.0)).flaeche(), 2.0)
        ↪  == 0;
        assert Double.compare((new Linie(-2.0, -2.0, -2.0, -2.0)).flaeche(),
        ↪  0.0) == 0;
    }
}
```

1. Refactoring - Replace Magic Number

```
public class Linie {
    private static final double TWO = 2.0;
    private double startX;
```

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

2. Refactoring - Introduce Parameter Object

```
1   public class Linie {
2
3       public static class Point {
4           public final double x;
5           public final double y;
6
7           public Point(double x, double y) {
8               this.x = x;
9               this.y = y;
10          }
11      }
12
13      private static final double TWO = 2.0;
14      private Point start;
15      private Point end;
```

```
16
17      public Linie(double startX, double startY, double endX, double endY) {
18          this.start = new Point(startX, startY);
19          this.end = new Point(endX, endY);
20      }
21
22      public Linie(Point start, Point end) {
23          this.start = start;
24          this.end = end;
25      }
26
27      public double distanz() {
28          return Math.sqrt(Math.pow(end.x - start.x, TWO) + Math.pow(end.y -
            ↪  start.y, TWO));
29      }
30
31      public double flaeche() {
32          return (Math.abs(end.x - start.x) * Math.abs(end.y - start.y)) / TWO;
33      }
34  }
```

Mit Anpassung der Testfunktion:

```
1   private static void testConstructor() {
2      Linie[] cases = {
3               new Linie(0.0, 0.0, 0.0, 0.0),
4               new Linie(-1.0, -1.0, -1.0, -1.0),
5               new Linie(Double.MAX_VALUE, Double.MIN_VALUE, Double.MIN_NORMAL,
                ↪  -Double.MAX_VALUE),
6               new Linie(new Linie.Point(0.0, 0.0), new Linie.Point(0.0, 0.0)),
7               new Linie(new Linie.Point(-1.0, -1.0), new Linie.Point(-1.0,
                ↪  -1.0)),
8               new Linie(new Linie.Point(Double.MAX_VALUE, Double.MIN_VALUE),
9                     new Linie.Point(Double.MIN_NORMAL, -Double.MAX_VALUE))
10     };
11     for (Linie linie : cases) {
12        assert linie != null;
13     }
14  }
```