RUHR
UNIVERSITÄT
BOCHUM

RUB

Software Engineering

Übung 1 - WS 2022/23

Group H7

Jasper Angl (108021103663)
Philipp Lehmann (108021228860)
Malte Janek Kottmann (108021220217)

Dozent des Kurses
Prof. Dr. Thorsten Berger

November 2022

1 Assignment 4

1.1 Prinzip: Information hiding

Information hiding ist ein essenzieller Bestandteil der guten Code ausmacht.Auch wir haben versucht dieses Prinzip zu integrieren. Besonders gut zu sehen ist dies in der main.java.

```
1  public class Main {
2
3      public static void main(String[] args) throws
4          IOException {
5
6          Apartment apartment = createApartment();
7          apartment.enter();
8      }
9
10     private static Apartment createApartment() {
11         Apartment apartment = new Apartment();
12         Room workroom = new Room("Workroom", false);
13         Room bedroom = new Room("Bedroom", false);
14         Kitchen kitchen = new Kitchen
15             ("Kitchen", false, false);
16         Bathroom bathroom = new Bathroom
17             ("Bathroom", false, false);
18         apartment.extension(workroom);
19         apartment.extension(bedroom);
20         apartment.extension(kitchen);
21         apartment.extension(bathroom);
22         return apartment;
23     }
24 }
```

Anmerkung: Formatierung stimmt nicht mit tatsächlicher Code Formatierung überein und ist aus optischen Gründen angepasst worden.
Die Klasse `Main` ist öffentlich zugänglich, sowie die `main()` Methode in der die Apartment-tour gestartet wird. Vor allem innerhalb dieser Methode sollte so nur so viel Code wie notwendig verwendbar und kein unterliegender Code manipulierbar sein. Grundsätzlich gilt: Nur was auch öffentlich sein muss, sollte öffentlich gemacht werden. Dementsprechend haben wir alle Variablen als `private` definiert, wie im Folgebeispiel zu sehen ist. Nur bei Code der innerhalb anderer Unterklassen verwendet werden soll wurde der Access-Modifier protected verwendet.

```
1  public class Room {
2      private String name;
3      private boolean lightning;
4      private ArrayList<Room> neighbors;
5      public Room(String name, boolean lightning) {
6          this.name = name;
```

```
7           this.lightning = lightning;
8           this.neighbors = new ArrayList<>();
9       }
```

1.2 Prinzip: modularity

Modularität bietet viele positive Aspekte, die je nach Ansatz verschieden deutlich hervortreten. Der bekannteste ist der Top-Bottom Ansatz. Bei diesem wird ein großes Problem in kleiner Probleme zerlegt, welche falls nötig wieder in kleiner Teilprobleme zerlegt werden. So lange bis Probleme gefunden wurden die Lösbar sind. Es ist wichtig, dabei darauf zu achten, dass bereits ausgearbeitete Lösungen kleinerer Teilprobleme eventuell mehrfach eingesetzt werden können. Aus dieser Allgemeingültigen Lösung ergibt sich dann die Modularität einzelner Komponenten.

```
1       protected int receiveReply(int range) {
2           Scanner reader = new Scanner(System.in);
3           int answer = reader.nextInt();
4           if (answer > 0 && answer <= range) {
5               return answer;
6           } else {
7               throw new IllegalArgumentException
8               ("Wrong integer input");
9           }
10      }
```

Die Funktion `receiveReply()` ist für die User Interface Eingabe Verarbeitung zu ständig und wird von der enter() Funktionen aus mehreren Unterklassen des Codes aufgerufen. Da sie allgemeingültig funktioniert, kann sie als Modul beliebig oft implementiert werden, um diese spezielle Aufgabe zu erfüllen.