1)

Ein Online-Lieferdienst für Lebensmittel führt ein Bonuspunktesystem ein. Bestellungen sind als registrierter sowie als anonymer Kunde möglich. Registrierte Kunden erhalten ab einer Einkaufssumme von ≥ 10€ nach dem Einkauf zwei Bonuspunkte gutgeschrieben. Ab zehn Bonuspunkten werden diese automatisch eingelöst und die Einkaufssumme wird um 2€ reduziert. Definieren sie anhand der gegebenen Informationen alle Äquivalenzklassen. Und geben sie zwei wichtige Testfälle (nicht JUnit-Tests) auf Basis von Grenzwerten an.

2)

Kann für die Klasse `MoneyTransferService` ein Integrationstest mit Hilfe von Mocking durchgeführt werden? Begründen Sie Ihre Antwort ausführlich.

(Vorlesungswiki Notiz: die Methode `setBankServicee` hat den Schreibfehler auch so in der Originalangabe)

```
1  public class MoneyTransferService {
2
3      private BankService bs;
4
5      public void setBankServicee(BankService bs) { this.bs = bs; }
6
7      public MoneyTransferResult transferMoney(String ibanFrom, String ibanTo, int amount)  {
8          this.bs = new SimpleBankService();
9          if(!isValid(ibanFrom))
10             throw new InvalidIbanException("Invalid from IBAN");
11         if(!isValid(ibanTo))
12             throw new InvalidIbanException("Invalid to IBAN");
13         if(amount <= 0)
14             throw new InvalidAmountException("Amount must be positive");
15         return bs.transfer(ibanFrom, ibanTo, amount);
16     }
17
18 }
```

3)

Führen Sie ein Review der nachfolgenden (JUnit 4) Testfälle durch.

```
1  public class SetTest {
2
3      private static Set<Integer> integersSet;
4
5      @BeforeClass
6      public static void setUp() {
7         integersSet = new HashSet<>();
8         integersSet.add(1);
9         integersSet.add(2);
10        integersSet.add(3);
11     }
12
13     @Test
14     public void testSetDoesSomethingShouldFail() {
15         assert(!integersSet.remove(3));
16     }
17
18     @Test
19     public void setRemoveShouldReturnTrueIfElementIsRemoved() {
20         for(int i = 1; i <= 3; i++) {
21             if(!integersSet.remove(i))
22                 throw new IllegalStateException("Failed to remove value from set");
23         }
24     }
25
26     @Test(expected = IllegalStateException.class)
27     public void addingExistingElementToSetShouldEnsureElementExistsOnlyOnce() {
28         integersSet.add(1);
29         integersSet.remove(1);
30         if(!integersSet.remove(1))
31             throw new IllegalStateException("Failed to remove value from set");
32     }
33 }
```

Finden Sie fünf Fehler in der Implementierung der Testfälle und erklären Sie die gefundenen Probleme und geben Sie Vorschläge zur Behebung.
Achten Sie im Speziellen auf Testing Bad Practices!
Hinweis: Sollte ein Fehler mehrfach vorkommen, zählt dieser nur als ein Fehler.

Zeilen	Beschreibung

4)

Bei der folgenden Klasse `AlarmService` handelt es sich um ein Service das zwischen einer definierten Startzeit und Endzeit eine Alarmanlage aktiviert. testen Sie die Methode `isAlarmActive(int startHour, int endHour, boolean daylightSaving)` auf ihre Korrekte Funktionsweise, indem Sie folgende Testfälle in JUnit-Syntax mit mocking implementieren.
(Verwenden Sie kein Mocking Framework!).

a) `startHour = 20, endHour = 6, daylightSaving = false; result: true`
b) `startHour = 8, endHour = 16, daylightSaving = true; result: false`

```
interface Clock {
    // throws IllegalArgumentException if timeZoneOffset is not in range [+-24]
    Integer getDayTimeInMillis(Integer timeZoneOffset) throws IllegalArgumentException;
}

class AlarmService {
    private Clock clock;
    public boolean isAlarmActive(int startHour, int endHour, boolean daylightSaving) {
        Integer dayTime = clock.getDayTimeInMillis(daylightSaving ? 1 : 0) / 1000 / 60 / 60;
        if(startHour <= endHour) return (dayTime >= startHour) && (dayTime <= endHour);
        else return (dayTime <= startHour) && (dayTime >= endHour);
    }
    public void setClock(Clock clock) { this.clock = clock; }
}
```

5)

Implementieren Sie folgende vorgegebenen Tests in JUnit

a) `removeCorrectValueForGivenKey();`
Überprüft ob für den richtigen Value der richtige Key zurückgegeben und aus der Map entfernt wird.

b) `removeShouldFailWhenKeyDoesNotExist();`
Überprüft ob eine `NoSuchElementException` beim Aufruf von `remove` geworfen wird, wenn der Key nicht existiert.

c) `removeAllShouldRemoveAllElements();`
Überprüft ob alle Elemente aus der Map entfernt wurden.

Ihre Aufgabe ist die Überprüfung der korrekten Funktionsweise der Methoden `remove(<key>)` und `removeAll()`. Die Methode `remove(<key>)` liefert den Wert am angegebenen Key und entfernt ihn aus der Map. Sollte die Map an der Stelle leer sein, wird eine `NoSuchElementException` geworfen. Die Methode `removeAll()` löscht alle Elemente. Zusätzlich steht Ihnen die sicher richtig implementierte Methode `size` zur Verfügung, die die Anzahl der Elemente in der Map zurückliefert.

```
 1 public class MapTest {
 2     private MyMap<String, Integer> map;
 3     private static final String NAME_1 = "Jane";
 4     private static final String NAME_2 = "John";
 5     private static final Integer AGE_1 = 25;
 6     private static final Integer AGE_2 = 26;
 7     @Before
 8     public void setUp() {
 9         map = MyMap<>();
10         map.put(NAME_1, AGE_1);
11         map.put(NAME_2, AGE_2);
12     }
```