Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Dr. J. Bendisposto

HEINRICH HEINE
UNIVERSITÄT
DÜSSELDORF

20. Juli 2019

Klausur

Professionelle Softwareentwicklung
Sommersemester 2019

Nachname: _____ Vorname: _____

Matrikelnummer: _____

Unterschrift: _____

Zugelassene Hilfsmittel: Eine handschriftlich beschriebene A4 Seite. Matrikelnummer und Name muss auf der Seite stehen

Sollten Sie zusätzliche Blätter benötigen, bekommen Sie diese von uns. Sie dürfen keine eigenen Blätter verwenden!

Diese Klausur enthält 10 nummerierte Seiten. Prüfen Sie bitte zuerst, ob alle Seiten vorhanden sind.

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	9	10	Σ
Punktzahl	3	7	7	10	10	10	8	10	10	15	90
Erreicht											

Aufgabe 1

[3 Punkte]

In einem Gradle-Projekt mit einer Standard-Konfiguration liegt eine Java-Datei unter dem Pfad `src/main/java/main/App.java`. Wie lautet die erste Zeile der Java-Datei, die vom Java-Compiler beachtet wird?

Aufgabe 2

[7 Punkte]

Schreiben Sie eine statische Methode `testUpperCaseList`, die eine Liste von Strings als Parameter übergeben bekommt und eine Liste zurückgibt, bei der die Strings in Großbuchstaben transformiert wurden. Der unten stehende Testfall ist ein Beispielaufruf. Ihr Code muss natürlich für jede Liste von Strings funktionieren. Verwenden Sie die Stream API. Für imperative Lösungen (z.B. mit for Schleifen, o.a.) gibt es keine Punkte.

```
@Test
public void testUpperCaseList() {
   List<String> eingabe = Arrays.asList("ProPra", "Klausur");
   List<String> resultat = StringListTransform.toUpperCaseList(eingabe);
   List<String> expected = Arrays.asList("PROPRA", "KLAUSUR");
   assertEquals(expected, resultat);
}
```

Tipp: Verwenden Sie die Methode `toUpperCase` der Klasse `String`.

Beispielaufruf: `"foo".toUpperCase()` gibt `"FOO"` zurück.

Aufgabe 3

[7 Punkte]

Nennen Sie zwei Qualitätsanforderungen, die sich widersprechen können. Beschreiben Sie die Anforderungen und demonstrieren Sie an einem Beispiel wie sie sich widersprechen können.

Aufgabe 4

[10 Punkte]

Teil eines Korrektursystems ist die folgende Klasse, die Abgaben repräsentiert. Überprüfen Sie die Klasse im Hinblick auf die SOLID Prinzipien. Erklären Sie, ob und welche Prinzipien verletzt wurden. Begründen Sie Ihre Antwort und erklären Sie, wie der Code verbessert werden müsste.

```
@Getter // Erzeugt Getter für alle Attribute
public class Abgabe {
    private List<Datei> dateien = new ArrayList<>();
    private int punkte = 0;
    private boolean korrigiert = false;

    public void dateiHinzufuegen(Datei datei) {
        dateien.add(datei);
    }

    public void korrekturAbschliessen(int punkte) {
        this.punkte = punkte;
        this.korrigiert = true;
    }

    public void zuordnen(Korrektor k) {
        k.addAbgabe(this);
    }
}
```

Aufgabe 5

[10 Punkte]

Gegeben sei folgende Klasse, die mittels Interpolationssuche[1] effizient feststellen kann, ob ein Wert in einer Sammlung von int-Werten enthalten ist. Die Klasse hat ein Problem, das dazu führen kann, dass die Interpolationssuche kaputt geht. Erklären Sie, was genau das Problem verursacht und schlagen Sie eine Lösung vor.

```
public class Interpolation {
    private final int[] array;

    public Interpolation(int[] eingabe) {
        Arrays.sort(eingabe); // sortiert eingabe in place
        this.array = eingabe;
    }

    public boolean contains(int wert) {
        // Interpolationssuche
        // Details sind egal, benötigt sortiertes Array
    }
}
```

[1] Eine Interpolationssuche ist eine Variante der binären Suche auf einer sortierten Sammlung von Werten. Es wird aber nicht immer in der Mitte geteilt, sondern eine geeignete Stelle "geraten". Stellen Sie sich vor, Sie müssten das Wort Ünterhaltungïn einem Wörterbuch nachschlagen. Sie würden vermutlich das Wörterbuch irgendwo am Ende aufschlagen und nicht in der Mitte.

Aufgabe 6

[10 Punkte]

Gegeben sei folgender Code. Schreiben Sie den Code so um, dass er das *Tell, don't ask*-Prinzip respektiert.

```
public class Monitor {
    public void sendeAlarm(String msg) { /* ... */ }
    public int getSystemTemperatur() { /* ... */ }
}

public class RechnerKontrolle {
    public void healthCheck(Monitor monitor) {
        if (monitor.getSystemTemperatur() > 93) {
            monitor.sendeAlarm("CPU überhitzt");
        }
    }
}
```

Aufgabe 7

[8 Punkte]

Gegeben ist folgender HTML Schnipsel, der von einer Anwendung im Rahmen eines Userinterfaces ausgeliefert wurde. Der Code hat ein sehr ernstes Problem. Identifizieren und erklären Sie das Problem und erläutern Sie mögliche Konsequenzen, die daraus resultieren.

```
<table>
<tr>
 <td>Name</td>
 <td>Actions</td>
</tr>
<tr>
 <td>Andreas Schmidt</td>
 <td>
   <a href="details/1">Userprofil anzeigen</a>
   <a href="delete/1">Löschen</a>
 </td>
</tr>
<tr>
 <td>Carola Müller</td>
 <td>
   <a href="details/2">Userprofil anzeigen</a>
   <a href="delete/2">Löschen</a>
 </td>
</tr>
</table>
```

Aufgabe 8

[10 Punkte]

Folgender Test wurde für Projekt 4 geschrieben. Was würden Sie an dem Test kritisieren und warum?

Beschreiben Sie, wie der Test aussehen müsste, so dass er den in der Vorlesung diskutierten Prinzipien für gute Tests entspricht.

```
@Test
public void einKorrektor_nachtraeglicheAbgabe() {
  Abgabe abgabe1 = new Abgabe();
  Blatt blatt = new Blatt(0, asArrayList(abgabe1));
  when(blattService.getBlatt(0)).thenReturn(blatt);
  when(korrektorService.getAll()).thenReturn(singleKorrektorTemplate());

  ZuordnungsService zuordnungsService =
       new ZuordnungsService(blattService, korrektorService);
  zuordnungsService.abgabenZuordnen(0);
  assertNotNull(abgabe1.getKorrektor());

  // Nachträgliche Abgabe
  Abgabe abgabe2 = new Abgabe();
  blatt.addAbgabe(abgabe2);

  zuordnungsService.abgabenZuordnen(0);
  assertNotNull(abgabe2.getKorrektor());
  assertSame(abgabe1.getKorrektor(), abgabe2.getKorrektor());
}
```

Aufgabe 9

[10 Punkte]

In einem Projekt haben wir einzelne Tests, die wichtig, aber leider verhältnismäßig langsam sind. Es wurde alles versucht, die Tests zu beschleunigen, aber einige Tests benötigen immer noch einige Minuten. Wir wollen unsere Tests daher so gruppieren, dass langsame Tests nicht immer ausgeführt werden.

Ändern Sie den beispielhaften langsamen Test so ab, dass das Testresultat von JUnit **ignoriert** wird, wenn die Methode `runSlowTests()` den Wert `false` liefert.

```
@Test
public void slowTest() {
    int result = langsamerCode();
    assertThat(result, is(42));
}
```

Aufgabe 10

[15 Punkte]

Wir wollen eine Webanwendung zur Addition von ganzen Zahlen schreiben. Dazu verwenden wir folgendes Formular, das aus der Datei `form.html` stammt:

```
<h1>Summe</h1>
<form th:object="${summe}" method="POST">
    <input type="number" th:field="*{a}">
    <input type="number" th:field="*{b}">
    <input type="submit">
</form>
<p>Ergebnis: <span th:text="${summe.summe}">50</span></p>
```

Zusätzlich haben wir eine Java-Bean, die das Formular in unserer Webanwendung repräsentiert. Wir verwenden die Lombok Annotation `@Data`, die automatisch Getter und Setter für alle Felder der Klasse, sowie den Konstruktor und einige Hilfsmethoden (equals, toString, ...) generiert.

Ergänzen Sie die Datenklasse und den Controller so, dass die Anwendung die Summe der eingegebenen Zahlen berechnet und anzeigt. Sie brauchen keine Fehlerbehandlung zu implementieren (z.B. wenn Texte statt Zahlen eingegeben werden).

Das Codetemplate für den Controller befindet sich auf der nächsten Seite

```
@Data
public class Summe {
  private int a,b;
  // Ergänzen Sie hier




}
```

```
@Controller
public class SummenController {

   @GetMapping("/")
   // Ergänzen Sie hier









   @PostMapping("/")
  // Ergänzen Sie hier









}
```