HHU Düsseldorf
Institut für Informatik
Dr. J. Bendisposto
Dr. M. Brenneis

Daten des Prüflings

Matrikelnummer: _____

Nachname: _____

Vorname: _____

2. Klausur

Programmierpraktikum I
23. September 2024

- Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält.
- Sie dürfen auf den leeren Rückseiten schreiben. Bei Bedarf erhalten Sie von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie unten auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Erlaubte Hilfsmittel: eine beidseitig beschriebene A4-Seite, Wörterbuch (Letzteres muss vor Klausurbeginn der Aufsicht zur Kontrolle vorgelegt werden.)
- Schalten Sie technische Geräte aus. Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

☐ Falls ich diese Klausur bestehe, möchte ich **nicht** automatisch zum Programmierpraktikum 2 im kommenden Wintersemester angemeldet werden.

Zusätzliche Blätter: _____ **Wir wünschen Ihnen viel Erfolg!**

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	Σ
Punkte	9	7	2	2	3	3	26
Erreicht							

Aufgabe 1

_____ / 9 Punkte

Wir wollen etwas Marketing betreiben. Um gezielt auf unsere Kundschaft zuzugehen, wollen wir die Namen der drei Kund:innen mit dem niedrigsten Umsatz (= Summe der Werte aller Einkäufe einer Person) finden. Die uns zur Verfügung stehenden Daten sind durch folgende Klassen definiert:

```
1 public record Einkauf(LocalDate datum, int wert) {}
2 public record Kunde(String name, List<Einkauf> bestellungen) {}
```

In der gesamten Aufgabe müssen Sie **Streams** verwenden, wo es möglich ist. Sie können Lösungen aus vorangehenden Aufgabenteilen verwenden, selbst wenn Sie diese Teilaufgabe nicht gelöst haben. Die folgenden drei Methoden sollen alle in `Kunde` liegen.

___/4 (a) Schreiben Sie eine Objekt-Methode, die den Umsatz für ein:e Kund:in berechnet. Verwenden Sie Methodenreferenzen, wo immer es möglich ist.

Zur Erinnerung: Die generische `Stream<T>` Klasse besitzt **keine** Methode `sum`.

```
public int umsatz() {
  return
```

```
bestellungen.stream()
.mapToInt(Einkauf::wert)
.sum();
```

```
}
```

___/3 (b) Schreiben Sie eine Methode `umsatzVergleich`, die eine Implementierung des funktionalen Interfaces `Comparator` zurückliefert. Der `Comparator` soll in Aufgabenteil c) benutzt werden, um `Kunde`-Objekte anhand ihres Umsatzes zu sortieren.

```
public static Comparator<Kunde> umsatzVergleich() {
  return
```

```
(k1, k2) -> Integer.compare(k1.umsatz(), k2.umsatz())
```

```
}
```

___/2 (c) Vervollständigen Sie die Methode, die die **Namen** der drei Kund:innen mit dem **kleinsten** Umsatz berechnet.

```
public static List<String> bottom3(List<Kunde> kunden) {
  return kunden.stream()
```

```
    .sorted(umsatzvergleich())
    .map(Kunde::name)

  .limit(3)
  .toList();
```

```
}
```

Aufgabe 2

_____ / 7 Punkte

Wir wollen die Software für ein System zur Qualitäts-Kontrolle in einer Fabrik entwickeln. Die Hardware des Systems besteht aus einem Sensor, der die hergestellten Teile fotografiert, und einem Luftausstoß-System, das fehlerhafte Teile mit einem Luftstoß vom Fließband pustet.

Folgender Code ist bereits entwickelt worden:

```
1  public class QAImage {
2    public boolean qualitaetOK() {
3       // Implementierung nicht abgedruckt
4       // true <=> Qualität OK
5    }
6  }
7  public interface Sensor { QAImage nextPicture(); }
8  public interface Luftausstoss { void pusten(); }
9  public class Sorter {
10   private Sensor sensor;
11   private Luftausstoss ausstoss;
12   public Sorter(Sensor sensor, Luftausstoss ausstoss) {
13     // Zuweisung der Parameter in die privaten Attribute nicht abgedruckt
14   }
15 }
```

Testgetrieben soll in der Klasse `Sorter` eine Methode `void pruefen()` entwickelt werden, die das nächste gefertigte Teil fotografiert, prüft, ob die Qualität das Teils OK ist, und es gegebenenfalls vom Fließband pustet (wenn die Qualität *nicht* OK ist).

___/5 (a) Vervollständigen Sie die Testmethode, die den Fall prüft, dass die Qualität des nächsten Teils OK ist.

___/2 (b) Geben Sie für **alle** Instanzen vom Typ `QAImage`, `Sensor`, `Luftausstoss`, `Sorter`, die Sie im Test verwenden, an, ob es sich um ein Dummy, Stub, Mock oder um das echte Objekt handelt.

```
@Test
@DisplayName("Das nächste Teil ist OK und wird nicht weggepustet")
void pruefeTest1() {


        // Arrange
        QAImage img = mock(QAImage img); // Stub
        when(img.qualitaetOK()).thenReturn(true);

        Sensor sensor = mock(Sensor.class); // Stub
        when(sensor.nextPicture()).thenReturn(img);

        Luftausstoss ausstoss = mock(Luftausstoss.class); // Mock

        Sorter sorter = new Sorter(sensor, luftausstoss); // echtes Objekt

        // Act
        sorter,pruefen();

        // Assert
        verify(luftausstoss, never()).pusten();



}
```

Aufgabe 3

_____ / 2 Punkte

Im ersten Programmierprojekt haben wir eine Anwendung geschrieben, bei der Gruppen von Personen, die gemeinsam Ausgaben tätigen, fair die Kosten verteilen können. Folgender Code zeigt einen Ausschnitt aus einer möglichen (funktionierenden) Lösung:

```
1  record Ueberweisung(String zahler, String empfaenger, double betrag) { }
2
3  class Berechnung {
4    static Collection<Ueberweisung> ueberweisungenBerechnen(final Map<String, Double> ausgaben) {
5
6      // Implementierung der aufgerufenen Methoden in den drei folgenden Zeilen ist irrelevant.
7      // Diese nicht abgedruckten Methoden sind alle pure.
8      Set<String> personenDieGeldBekommen = hatMehrAlsSchnittAusgegeben(ausgaben);
9      Set<String> personenDieGeldZahlen = hatWenigerAlsSchnittAusgegeben(ausgaben);
10     double mittlereAusgaben = mittlereAusgaben(ausgaben);
11
12     List<Ueberweisung> ueberweisungen = new ArrayList<>();
13
14     for(var zahler: personenDieGeldZahlen) {
15       ausgaben.replace(zahler, mittlereAusgaben - ausgaben.get(zahler));
16       for(var empfaenger: personenDieGeldBekommen) {
17         if(ausgaben.get(zahler) > 0 && ausgaben.get(empfaenger) > mittlereAusgaben) {
18           double zahlung = Math.min(ausgaben.get(empfaenger) - mittlereAusgaben,
               ausgaben.get(zahler));
19           ausgaben.replace(empfaenger, ausgaben.get(empfaenger) - zahlung);
20           ausgaben.replace(zahler, ausgaben.get(zahler) - zahlung);
21           ueberweisungen.add(new Ueberweisung(zahler, empfaenger, zahlung));
22         }
23       }
24     }
25     return ueberweisungen;
26   }
27 }
```

___/1 (a) Geben Sie **begründet** an, ob Tests für die Methode `ueberweisungenBerechnen` so geschrieben werden können, dass sie das R in FIRST erfüllen.

Ja, wir können problemlos Tests schreiben, die repeatable sind.
Die Methode hängt nur von Ihren Eingaben ab und ist deterministisch

___/1 (b) Ist die Methode seiteneffektfrei? Begründen Sie Ihre Antwort.

Nein, die Methode ist nicht seiteneffektfrei.
Beispielsweise in Zeile 15 wird der übergebene Parameter geändert.

Aufgabe 4

_____ / 2 Punkte

Wir haben einen Teil einer Spring-Anwendung zur Verwaltung von Bestellungen entwickelt.

```
1  public interface Order {
2    public void markDelivered();
3  }
4
5  public class SmallOrder implements Order {
6    // relevanter Ausschnitt der Implementierung:
7    private boolean delivered = false;
8    public void markDelivered() { delivered = true; }
9  }
10
11 @Component
12 public class OrderRepository {
13   public Order findOrderById(Long id) {
14     // Implementierung ausgelassen
15   }
16
17   public void save(Order order) {
18     // Implementierung ausgelassen
19   }
20 }
21
22 @Component
23 public class OrderService {
24
25   private final OrderRepository repository;
26
27   public OrderService(OrderRepository repository) {
28     this.repository = repository;
29   }
30
31   public void deliver(Long orderId) {
32     Order order = repository.findOrderById(orderId);
33     order.markDelivered();
34     repository.save(order);
35   }
36 }
```

Untersuchen Sie den gegebenen Code im Hinblick auf Verletzungen der Prinzipien DIP, LSP, ISP und OCP. Kreuzen Sie dazu in der Tabelle auf der nächsten Seite an, ob das jeweilige Prinzip verletzt ist. Ein Prinzip gilt hier nur dann als verletzt, wenn Sie die Verletzung im gegebenen Code sehen können.

Geben Sie für jedes verletzte Prinzip an, woran die Verletzung erkennbar ist. Verwenden Sie ausschließlich Informationen, die Sie dem vorgegebenen Code entnehmen können.

	verletzt?	Begründung (**nur wenn *ja* angekreuzt**)
DIP	☒ ja ☐ nein	OrderService verwendet direkt die Klasse OrderRepository anstelle eines Interfaces.
ISP	☐ ja ☒ nein	
LSP	☐ ja ☒ nein	
OCP	☐ ja ☒ nein	

Aufgabe 5

_____ / 3 Punkte

Für eine Arztpraxis soll ein Verwaltungssystem für Patient:innen entwickelt werden. In dem Package `de.propra.patient` haben wir dazu folgende Klassen geschrieben. Um den Code kurz zu halten, haben wir die Id-Attribute und die `equals`- und `hashcode`-Methoden nicht abgedruckt.

```
1  record Termin(LocalDateTime datum, String behandlung) {}
2
3  class Stammdaten {
4    private String name;
5    private String adresse;
6    public String getName() { return name;}
7    public void setName(String name) { this.name = name;}
8    public String getAdresse() { return adresse;}
9    public void setAdresse(String adresse) { this.adresse = adresse;}
10 }
11
12 // AggregateRoot
13 public class Patient {
14   private final Stammdaten stammdaten;
15   private final ArrayList<Termin> termine;
16
17   public Patient(String name, String adresse, ArrayList<Termin> termine) {
18     this.termine = termine;
19     this.stammdaten = new Stammdaten();
20     this.stammdaten.setName(name);
21     this.stammdaten.setAdresse(adresse);
22   }
23
24   public void namenAendern(String neuerName) { stammdaten.setName(neuerName); }
25
26   public String name() { return stammdaten.getName(); }
27
28   public void neuerTermin(LocalDateTime datum, String behandlung) {
29     if(termine.size() < 4) {
30       termine.add(new Termin(datum, behandlung));
31     }
32   }
33
34   public String zuletztVereinbarterTermin() { return termine.getLast().toString(); }
35 }
```

___/1 (a) Wir haben einen Fehler bei der Implementierung bezüglich der **Regeln für Aggregate** gemacht. Identifizieren Sie die genaue **Stelle**, an der der Fehler ist, und **begründen** Sie, warum die Stelle fehlerhaft ist.

Das Problem ist der Parameter termine, der im Konstruktor übergeben wird. Die übergebene ArrayList ist veränderbar und wird direkt übernommen. Dadurch könnte die Liste später von außen geändert werden, ohne dass das Aggregat das verhindern kann.

Wir hätten hier im Konstruktor eine Kopie der Liste anfertigen müssen.

___/2 (b) Angenommen, wir wollen das System vor Problemen bezüglich der Nebenläufigkeit mithilfe des `synchronized`-Keywords absichern. Welche Methoden in welchen Klassen müssen wir `synchronized` deklarieren? Wir wollen dabei möglichst wenig `synchronized` verwenden. Begründen Sie Ihre Antwort in 2–3 Sätzen.

Wir müssen unbedingt die Methode neueTermine synchronisieren, weil es dort zu einer Invariantenverletzung kommen kann. Das würde z.B. passieren, wenn wir schon 3 Termine haben und zwei Threads gleichzeitig neuerTermin aufrufen. Beie Threads lesen zuerst termine.size() und erhalten den Wert 3, dann können beide Threads jeweils einen Termin hinzufügen und wir haben 5 Termine gespeichert.

Aufgabe 6

_____ / 3 Punkte

Wir haben folgendes Gradle-Projekt aufgesetzt:

Verzeichnis-Struktur:

```
|-- build.gradle
|-- src
    |-- main
        |-- java
            |-- Main.java
```

Inhalt der `build.gradle`:

```
plugins {
  id 'java'
  id 'application'
}
application {
  // korrekte Konfiguration der Main-Klasse
  // nicht abgedruckt
}
repositories {
  mavenCentral()
}
dependencies {
  implementation 'com.google.guava:guava:33.1.0-jre'
}
```

Die Klasse `Main` benutzt die Klasse `ArrayListMultimap` aus der Guava-Bibliothek, die in der `build.gradle` angegeben ist. Das Projekt kann erfolgreich mit `gradle build` kompiliert werden und es wird eine jar-Datei `build/libs/helloworld.jar` erzeugt, die nur die `Main.class` und ein Manifest enthält.

___/1 (a) In welchem Package liegt die Klasse `Main`?

Unnamed Package (a.k.a Default Package)

___/1 (b) Wenn wir versuchen, das Programm wie folgt zu starten, gibt es einen Fehler:

```
% java -cp build/libs/helloworld.jar Main
Exception in thread "main" java.lang.NoClassDefFoundError: com/google/common/collect/ArrayListMultimap
        at Main.main(Main.java:5)
Caused by: java.lang.ClassNotFoundException: com.google.common.collect.ArrayListMultimap
        at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
        at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:526)
        ... 1 more
```

Erklären Sie in 1–2 Sätzen, was die **Ursache** der Fehlermeldung ist.

Der Klassenpfad enthält die benötigte Guava-Bibliothek nicht.

___/1 (c) Wie können wir das Programm mithilfe von Gradle (fehlerfrei) laufen lassen?

gradle run