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

Daten des Prüflings

Matrikelnummer: _____

Nachname: _____

Vorname: _____

1. Klausur

Programmierpraktikum I
24. Juli 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!

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

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	Σ
Punkte	9	8	2	2	2	4	27
Erreicht							

Aufgabe 1

_____ / 9 Punkte

Im Fachschaftsraum soll auf einem Monitor angezeigt werden, welche Busse und Bahnen **als Nächstes** an den Haltestellen Universität Ost und Universität Mitte abfahren. Es sollen immer **maximal** `nStueck` Bahnen, **sortiert nach Abfahrtszeit** aufgelistet werden. Die Abfahrtszeiten aller Fahrten werden von der Webseite des VRR mithilfe der Klasse `VrrApi` abgefragt; die Webseite liefert teilweise aber auch Abfahrten, die in der **Vergangenheit** liegen und nicht mit angezeigt werden sollen.

Wir haben schon etwas Code geschrieben:

```
public record Abfahrt(String linie, String ziel, LocalDateTime zeit)
  implements Comparable<Abfahrt> {
    public int compareTo(Abfahrt other) { return zeit.compareTo(other.zeit); }
}
```

```
public class VrrApi {
    public Stream<Abfahrt> getAbfahrten(String station) {
        // Implementierung der Abfrage
        // liefert endlich viele Ergebnisse
    }
}
```

```
public class AbfahrtMonitor {
    private static final List<String> STATIONEN = List.of("Uni Ost", "Uni Mitte");
    private final VrrApi vrr;

    public AbfahrtMonitor(VrrApi vrr) {
        this.vrr = vrr;
    }

    public List<Abfahrt> getNaechsteAbfahrten(int nStueck, LocalDateTime now) {
        return // Aufgabenteil b)
    }
}
```

___/2 (a) Wir haben Dependency Injection und Dependency Inversion kennengelernt. **Kreuzen** Sie in der Tabelle an, was davon im Code bezüglich der Klassen `AbfahrtMonitor` und `VrrApi` umgesetzt ist. **Erklären** Sie in jeweils einem Satz, woran im Code Sie Ihre Entscheidung festmachen.

	umgesetzt?	Begründung
Dependency Injection	☐ ja ☐ nein	
Dependency Inversion	☐ ja ☐ nein	

___/6 (b) Implementieren Sie die Methode `getNaechsteAbfahrten(int nStueck, LocalDateTime now)` mithilfe **genau eines** Stream-Ausdrucks. Fragen Sie dazu von allen `stationen` die Abfahrten ab und geben Sie die **nächsten** `nStueck` Abfahrten nach Abfahrtzeit sortiert zurück; alle Abfahrten, die zurückgegeben werden, müssen nach der Zeit `now` liegen. Verwenden Sie, wo es möglich ist, **Methodenreferenzen**.

Erinnerung: `LocalDateTime` hat eine Methode `boolean isAfter(LocalDateTime other)`.

```
public List<Abfahrt> getNaechsteAbfahrten(int nStueck, LocalDateTime now) {
    return




}
```

___/1 (c) Geben Sie für jeden Aufruf von **intermediären** Streamoperationen, bei denen sich der Typ der Elemente im Stream in Ihrer Lösung von (b) ändert, an, welchen Typ die Stream-Elemente vor und nach der Operation haben.

Beispiel:

```
IntStream.of(1,2,3)
    .boxed()        // int -> Integer
    .toList();
```

Aufgabe 2

_____ / 8 Punkte

Wir wollen die Methode `getNaechsteAbfahrten` in unserem Abfahrtsmonitor aus Aufgabe 1 natürlich auch testen.

Für unseren Test soll die Klasse `VrrApi` pro Haltestelle jeweils **genau eine** Abfahrt (also insgesamt zwei verschiedene Abfahrten) zurückgeben. Im Test soll geprüft werden, dass der Aufruf von `getNaechsteAbfahrten` **exakt** diese beiden Abfahrten zurückgibt. Wählen Sie für den Test passende Werte für die Parameter und die Abfahrten.

Erinnerung: Eine Instanz von `LocalDateTime` können Sie mit der Factory-Methode `of` erzeugen. Beispiel: 24.7.2024, 9:00 Uhr wird mit `LocalDateTime.of(2024,7,24,9,0)` erzeugt.

___/5 (a) Schreiben Sie die vollständige Testmethode, inklusive der zwingend nötigen Annotationen. Sie müssen keine Imports schreiben und können davon ausgehen, dass die üblichen Klassen und statischen Methoden importiert wurden.

__/3 (b) Wenn wir versuchen, die Klasse **`VrrApi`** zu testen, werden wir auf Probleme mit den FIRST-Prinzipien stoßen. **Kreuzen** Sie an, ob das Prinzip mit sehr hoher Wahrscheinlichkeit verletzt wird oder nicht. **Begründen** Sie bei den verletzten Prinzipien die Entscheidung in jeweils maximal zwei Sätzen.

	verletzt?	Begründung (nur wenn *ja* angekreuzt)
F	☐ ja ☐ nein	
I	☐ ja ☐ nein	
R	☐ ja ☐ nein	
S	☐ ja ☐ nein	
T	☐ ja ☐ nein	

Aufgabe 3

_____ / 2 Punkte

Eine Variante des Abfahrtmonitors soll als Teil einer Spring-Anwendung verwendet werden. Die neue Klassendefinition ist:

```java
public class SpringAbfahrtMonitor {

    private final List<String> stationen;
    private final VrrApi vrr;

    public SpringAbfahrtMonitor(List<String> stationen, VrrApi vrr) {
        this.vrr = vrr;
        this.stationen = stationen;
    }

    public List<Abfahrt> getNaechsteAbfahrten(int nStueck, LocalDateTime now) {
        // egal
    }
}
```

Genutzt wird die Klasse von einer anderen Spring-Bean, in die `SpringAbfahrtMonitor` automatisch per Konstruktor injiziert werden soll.

___/1 (a) Welche Änderung ist notwendig, damit `SpringAbfahrtMonitor` von Spring automatisch injiziert werden kann?

___/1 (b) Die Liste der Stationen soll aus einer Properties-Konfigurationsdatei von Spring injiziert werden. Der Dateiinhalt soll wie folgt aussehen:

`fsmon.haltestellen=Universität Ost,Universität Mitte`

Geben Sie an, wie der Konstruktor von `SpringAbfahrtMonitor` angepasst werden muss, damit die Haltestellen aus der Konfigurationsdatei injiziert werden.

Hinweis: Wenn die Property-Datei eine kommaseparierte Liste enthält, dann kann diese von Spring in eine Instanz von `List` injiziert werden.

Aufgabe 4

_____ / 2 Punkte

Für einen kleinen Online-Händler wollen wir den Wochentag für die Auslieferung einer Bestellung berechnen. Unsere erste Implementierung sieht folgendermaßen aus:

```
1  public enum Wochentag {
2    Mo,Di,Mi,Do,Fr,Sa,So
3  }
4
5  public class Versand {
6    Wochentag voraussichtlicherLiefertag(Wochentag bestelltag, boolean expressversand) {
7      if (expressversand) {
8        switch (bestelltag) {
9          case Mo:
10         case Di: return Mi;
11         case Mi: return Do;
12         case Do: return Fr;
13         default: return Di;
14       }
15     } else {
16       switch (bestelltag) {
17         case Mo:
18         case Di:
19         case Mi: return Fr;
20         default: return Mi;
21       }
22     }
23   }
24 }
```

Das Programm verletzt das OCP. Beschreiben Sie exakt, **woran** die Verletzung hier erkannt werden kann, und geben Sie an, **wie** die Verletzung behoben werden kann.

Aufgabe 5

_____ / 2 Punkte

Wir haben etwas Code geschrieben, um Personen in einem Versandsystem zu verwalten.

```java
public record Adresse(String strasse, String postleitzahl, String stadt) {}

public record Name(String vorname, String nachname) {}

public class Person {

  private final Name name;
  private final Adresse adresse;

  public Person(Name name, Adresse adresse) {
    this.name = name;
    this.adresse = adresse;
  }

  public String adressEintrag() {
    return String.format("""
    %s
    %s %s
    """, adresse.strasse(), adresse.postleitzahl(),adresse.stadt());
  }

  public void printVersandAufkleber() {
    System.out.println("An "+name.vorname()+" "+name.nachname());
    System.out.println(adressEintrag());
  }
}
```

___/1 (a) **Benennen** Sie den Code Smell, der in der Klasse `Person` vorliegt, und **erklären** Sie, wie Sie den Smell identifiziert haben.

___/1 (b) Wie könnten wir den Smell beseitigen?

Aufgabe 6

_____ / 4 Punkte

Wir haben aktuell den Branch `main` ausgecheckt. Bei der Ausführung des Befehls `git merge loesung` ist es zu einem Problem gekommen und wir haben folgende Meldung erhalten:

```
Auto-merging src/main/java/mocking/OptimizedAlgorithm.java
CONFLICT (content): Merge conflict in src/main/java/mocking/OptimizedAlgorithm.java
Automatic merge failed; fix conflicts and then commit the result.
```

Die betroffene Datei `OptimizedAlgorithm.java` sieht so aus:

```
1  package mocking;
2
3  public class OptimizedAlgorithm {
4
5    public static int[] pairedSum(int[] input) {
6      int[] result = new int[input.length / 2];
7      for (int i = 0; i < input.length; i += 2) {
8  <<<<<<< HEAD
9        result[i / 2] = input[i] + input[i + 1] + 0 * result[0]
10 =======
11       result[i / 2] = input[i] + input[i + 1] + 0 * result[0];
12 >>>>>>> loesung
13     }
14     return result;
15   }
16 }
```

___/1 (a) Wie könnte es zu dem Problem gekommen sein? Beschreiben Sie in 2–3 Sätzen, was auf welchem Branch passiert sein müsste, damit der Befehl die oben stehende Situation erzeugt.

___/1 (b) Wie müssen wir `OptimizedAlgorithm.java` ändern, sodass wir wieder eine lauffähige Version des Codes erhalten? Geben Sie an, welche Zeilen angepasst werden müssen.

___/2 (c) Nachdem wir `OptimizedAlgorithm.java` wie in Aufgabenteil (b) repariert haben, wollen wir die korrigierte Version nun in das Remote-Repository pushen. Geben Sie die dafür notwendigen Git-Befehle an.

```
git push
```