Hinweise zur Probeklausur

- Wir empfehlen, die Probeklausur unter so realistischen Bedingungen wie möglich zu schreiben. Das heißt, Sie sollten sich 45 Minuten möglichst **ungestört** mit der **ausgedruckten** Probeklausur beschäftigen und nur die für die echte Klausur erlaubten Hilfsmittel verwenden.

- Die Punkte pro Aufgabe entsprechen etwa dem Zeitaufwand in Minuten. Falls Sie bei einer Teilaufgabe nicht weiterkommen, überspringen Sie erstmal diese Teilaufgabe; ansonsten verlieren Sie in der echten Klausur unnötig Zeit.

- Sie dürfen bei jeder Aufgabe auf andere Methoden in der Aufgabe zurückgreifen, die sie selbst in der Aufgabe implementieren müssen oder die von uns vorgegeben sind. Ersteres ist auch dann erlaubt, falls sie diese Teilaufgabe nicht bearbeitet haben.

- Deckblatt, Hinweise usw. entsprechen voraussichtlich schon denen der echten Klausur. Sie können sich also bereits jetzt damit beschäftigen und alle Fragen dazu klären, dann müssen Sie das Deckblatt in der tatsächlichen Klausur nur noch überfliegen.

- Bei der echten Klausur bekommen Sie vorab einen Platz zugewiesen und auf diesem Platz liegt bereits eine Klausur, auf die Ihr Name gedruckt ist. Außerdem ist auf dem Deckblatt ein QR-Code, in dem Ihre Matrikelnummer steht. Wir nutzen den Code, um die Eintragung der Punkte in unsere Ergebnistabelle teilweise zu automatisieren.

- Die Klausur wird doppelseitig gedruckt und oben links getackert. Zusammenhängende Aufgaben werden dabei nach Möglichkeit (wie auch in der Probeklausur) beim Blättern nebeneinander zu liegen kommen.

- Fragen zur Probeklausur können Sie in den Tutorien, der Fragestunde oder im Ilias stellen. Außerdem gibt es voraussichtlich ein Lösungs-Video.

- In der Probeklausur können teilweise Abwandlungen alter Klausur- und Übungsaufgaben vorkommen. Dies kann in einer echten Klausur ebenfalls der Fall sein, muss es aber nicht.

- Bei jeder Klausur muss eine Auswahl aus den bearbeiteten Themen getroffen werden. Die Auswahl der Themen und die Verteilung der Punkte auf die einzelnen Themen kann in der echten Klausur anders aussehen. Welche Themen klausurrelevant sind, wird in der Vorlesung kommuniziert; insbesondere kennen die Hilfskräfte die Klausur nicht und können daher keine verbindlichen Aussagen zur Klausurrelevanz treffen.

- Wir bemühen uns, dass die Probeklausur eher **schwieriger** und zeitlich **knapper** kalkuliert ist als die echten Klausuren, und dass beide echten Klausuren gleich schwierig sind. Da der Schwierigkeitsgrad aber keine objektive Einschätzung ist, sondern von den eigenen Stärken und Schwächen abhängt, können wir das nicht garantieren. Die Klausuren orientieren sich an den im Semester bearbeiteten Materialien und den Lernzielen der Vorlesung. Die echten Klausuren sind keine Probeklausur „mit anderen Zahlen“.

- Zum weiteren Üben eigenen sich insbesondere die jüngeren Altklausuren, die Sie im Klausurarchiv der Fachschaft Informatik[1] finden. Beachten Sie, dass es in vergangenen Semestern Vorlesungsthemen gab, die dieses Jahr nicht behandelt wurden (und andersherum).

[1] https://fscs.hhu.de/de/klausurarchiv/

HHU Düsseldorf
Institut für Informatik
Prof. Dr. Michael Schöttner

Daten des Prüflings

Matrikelnummer: _____

Nachname: _____

Vorname: _____

Probeklausur

Programmierung WS 24/25
Dezember 2024

- Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält.
- Sie erhalten 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.
- Fachbegriffe werden wie in der Vorlesung definiert verwendet. Die Aufgaben beziehen sich auf die in der Vorlesung vorgestellte Java-Version 21. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Erlaubte Hilfsmittel: eine beidseitig beschriebene oder bedruckte A4-Seite, Wörterbuch (Wörterbuch 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	Σ
Punkte	6	12	6	3	6	33
Erreicht						

Aufgabe 1

_____ / 6 Punkte

___/2 (a) Sie haben eine Reihe von ganzzahligen Messwerten zeilenweise in der Textdatei `/projekt/werte.txt` gespeichert; ungültige Messwerte sind als -1 gespeichert. Außerdem haben Sie ein Java-Programm `Filter`, das zeilenweise Integer von der Standardeingabe einliest und alle Integer, die nicht -1 sind, wieder zeilenweise auf der Standardausgabe ausgibt.

```
/projekt % ls
Filter.class Filter.java werte.txt
/projekt %
```

Geben Sie einen Befehl an, mit dem Sie alle **gültigen** Messwerte aus `werte.txt` in einer Datei `bereinigt.txt` speichern können:

_____

___/4 (b) Gegeben sei die folgende Klasse:

Person.java — Java

```java
public class Person {

    private String name;
    private String mail;

    public Person(String name, mail) {
        this.name = name;
        this.mail = mail;
    }

    @Override
    public String toString() {
        return name + " " mail;
    }

}
```

Beim Compilieren der Klasse gibt es folgende Fehlermeldungen:

```
Person.java:6: error: <identifier> expected
    public Person(String name, mail) {
                                   ^
Person.java:13: error: ';' expected
        return name + " " mail;
                         ^
Person.java:13: error: not a statement
        return name + " " mail;
                          ^
3 errors
```

Geben Sie an, in welchen Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehler jeweils kurz (max. 1 Satz) und geben Sie die korrigierte Codezeile **vollständig** an, sodass der Code das tut, was bei der Programmierung wahrscheinlich vorgesehen war. Geben Sie keine Folgefehler an, die durch Korrektur eines vorherigen Fehlers behoben werden.

Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Aufgabe 2

_____ / 12 Punkte

Gehen Sie in dieser Aufgabe davon aus, dass $n \in \mathbb{N}$ und $n \geq 1$ gilt.

Eine natürliche Zahl n ist eine perfekte Zahl, wenn die Summe ihrer natürlichen Teiler (n ausgeschlossen) **gleich** n ist. Die kleinste perfekte Zahl ist 6 (Teilersumme $1 + 2 + 3 = 6$).

Unten ist ein Programm `PerfekteZahlen` mit einer Beispielausgabe vorgegeben. Erweitern Sie dieses Programm um folgende **private**, **statische** Methoden, damit das Programm wie im Beispiel funktioniert:

___/4 (a) Schreiben Sie eine Methode `int teilersumme(int n)`, welche die Summe aller natürlichen Teiler von `n` berechnet, wobei n selbst **ausgeschlossen** ist.

___/3 (b) Schreiben Sie eine Methode `boolean istPerfekt(int n)`, die genau dann `true` zurückgibt, wenn `n` eine perfekte Zahl ist.

___/5 (c) Schreiben Sie eine Methode `int[] perfekteZahlen(int anzahl)`, die ein Array zurückgibt, das genau die ersten `anzahl` perfekten Zahlen enthält.

Beispiel:

```
% java PerfekteZahlen
6
true
false
6 28
```

PerfekteZahlen.java — Java

```java
public class PerfekteZahlen {

    public static void main(String[] args) {
        System.out.println(teilersumme(6));
        System.out.println(istPerfekt(6));
        System.out.println(istPerfekt(12));

        for(int zahl: perfekteZahlen(2)) {
            System.out.print(zahl + " ");
        }
    }

    // Ergänzen Sie hier die Methoden teilersumme, istPerfekt und perfekteZahlen.
```

PerfekteZahlen.java (Fortsetzung) Java

```
}
```

Aufgabe 3

_____ / 6 Punkte

Aus der Vorlesung kennen Sie folgende Implementierung von Insertion Sort, die ein Array von Integern aufsteigend sortiert:

```java
public static void sort(int[] numbers) {
    for(int currentIndex = 0; currentIndex < numbers.length; currentIndex++) {
        int currentNumber = numbers[currentIndex];
        int insertionPosition = currentIndex;
        while(insertionPosition > 0 && numbers[insertionPosition - 1] > currentNumber) {
            numbers[insertionPosition] = numbers[insertionPosition - 1];
            insertionPosition--;
        }
        numbers[insertionPosition] = currentNumber;
    }
}
```

Zwei Strings können mithilfe der öffentlichen Instanzmethode `int compareTo(String other)` verglichen werden. Der Rückgabewert stellt hierbei das Ergebnis des Vergleichs wie folgt dar (`a` und `b` vom Typ `String`):

`a.compareTo(b) == 0`	→	`a` und `b` sind gleich
`a.compareTo(b) > 0`	→	`a` ist lexikographisch größer als `b`
`a.compareTo(b) < 0`	→	`a` ist lexikographisch kleiner als `b`

Vervollständigen Sie die Methode `sorted`, die ein Array von `String`-Objekten übergeben bekommt und ein neues Array zurückgeben soll, in dem dieselben Strings lexikographisch **absteigend** sortiert sind; die Reihenfolge der Strings im übergebenen Array soll von der Methode **nicht** verändert werden; das Original-Array und das sortierte Array sollen dieselben Objekte im Heap referenzieren. Sie dürfen davon ausgehen, dass der Methoden-Parameter `strings` ungleich `null` ist und keine Werte enthält, die `null` sind.

Ihre Lösung

```java
public static String[] sorted(String[] strings) {
```

Ihre Lösung (Fortsetzung) Java

```
}
```

Aufgabe 4

_____ / 3 Punkte

Newton möchte ein Programm schreiben, das über die Standardeingabe eine beliebige Anzahl von Unikennungen einlesen kann und anschließend alphabetisch sortiert. Er schlägt vor, die Unikennungen dafür in einem Array der Größe 200 zu speichern und anschließend zu sortieren.

Tracy sagt, dass er lieber eine einfach verkettete Liste verwenden sollte.

___/2 (a) Würden Sie eher Newtons oder Tracys Vorschlag umsetzen? Geben Sie einen nachvollziehbaren Grund in 1–3 ausformulierten Sätzen an.

___/1 (b) Newton erwidert, dass die Elemente einer verketteten Liste nicht mehr sortiert werden können, nachdem Sie einmal hinzugefügt worden sind. Stimmen Sie dieser Aussage von Newton zu? Begründen Sie Ihre Antwort in 1–2 ausformulierten Sätzen.

Aufgabe 5

_____ / 6 Punkte

In dieser Aufgabe sollen Sie Interfaces, Klassen und Methoden für eine Marketinganwendung eines Restaurants nach gewissen Vorgaben implementieren. Eine andere Person integriert dann später Ihre Klasse in die Gesamtanwendung.

Hinweise:

- Sie dürfen alle Variablennamen frei wählen.
- Wählen Sie sinnvolle Datentypen für Ihre Variablen.
- Alle Instanzvariablen müssen **privat** sein.
- Wenn kein Konstruktorverhalten vorgeschrieben ist, reicht der Default-Konstruktor.
- Innerhalb dieser Aufgabe müssen Sie keine Parameter validieren.

Gegeben ist folgendes Interface:

Weather.java — Java

```java
public interface Weather {
    // gibt die aktuelle Temperatur in Grad Celsius zurück
    double getTemperature();
}
```

___/2 (a) Schreiben Sie eine **nicht abstrakte, öffentliche** Klasse `FakeWeather`, die das `Weather`-Interface implementiert. Als aktuelle Temperatur soll irgendeine beliebige Zahl zurückgeben werden.

FakeWeather.java — Java

___/4 (b) Vervollständigen Sie die Klasse `Customer`. Jedes `Customer`-Objekt speichert eine E-Mail-Adresse und einen Namen, die beide dem **öffentlichen** Konstruktor als Parameter übergeben werden. Die zwei Eigenschaften sollen außerdem von **öffentlichen** Methoden `getMail()` bzw. `getName()` zurückgegeben werden.

Customer.java Java

```
public class Customer {









}
```