Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Dr. Markus Brenneis

22. März 2022

Nachklausur

Programmierung
WS 21/22

Nachname:		Vorname:	
Matrikelnummer:		Sitzplatznummer:	
Zusätzliche Blätter:		Unterschrift:	

Hinweise:

- Diese Klausur enthält 21 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- Sie erhalten außerdem 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 außerdem oben auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben. Wenn Sie weiteres Papier benötigen, melden Sie sich bitte.
- Alle Fachbegriffe in dieser Klausur werden wie in der Vorlesung definiert verwendet. Alle Fragen beziehen sich auf die in der Vorlesung vorgestellte Java-Version 11. 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.
- Zugelassene Hilfsmittel: eine beidseitig beschriebene oder bedruckte DIN-A4-Seite, Wörterbuch (Wörterbücher müssen vor Beginn der Klausur den Aufsichtspersonen zur Kontrolle vorgelegt werden.)
- Schalten Sie Ihr Mobiltelefon 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!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	9	Σ
Punktzahl	7	5	4	10	13	7	9	6	29	90
Erreicht										

Aufgabe 1

_____ / 7 Punkte

(a) [1 Punkt] Ihr Terminal sieht gerade wie folgt aus:

```
/mnt/projects/blatt03 % ls
Wind.java Wind.class
/mnt/projects/blatt03 %
```

Die Klasse `Wind` ist bereits kompiliert. Sie wollen die `main`-Methode der Klasse `Wind` mit dem Argument 5 ausführen. Welchen Befehl müssen Sie dazu eingeben?

(b) [4 Punkte] Gegeben sei die folgende Klasse:

Factorial.java — Java

```java
public class Factorial {
    public static int of(int n) {
        int result = 1;
        for(int i = 1; i < n; i++;) {
            result *= i;
        }}
        return result;
    }
}
```

Beim Compilieren der Klasse gibt es folgende Fehlermeldungen:

```
Factorial.java:4: error: ')' expected
        for(int i = 1; i < n; i++;) {
                                  ^
Factorial.java:4: error: illegal start of expression
        for(int i = 1; i < n; i++;) {
                                   ^
Factorial.java:7: error: illegal start of type
        return result;
        ^
Factorial.java:7: error: <identifier> expected
        return result;
                     ^
Factorial.java:9: error: class, interface, or enum expected
}
^
5 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: _____

Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

(c) [2 Punkte] Sie haben eine Datei `Punkt.java` im Ordner `/mnt/projekt/de/hhu/progra` gespeichert; der Classpath ist `/mnt/projekt/`. Vervollständigen Sie die ersten beiden Zeilen der Datei:

`Punkt.java` `Java`

```
_____ de._____;
public class _____ {
```

Aufgabe 2

_____ / 5 Punkte

Formen Sie die Kontrollstrukturen in den folgenden Codeausschnitten um, sodass sich die Semantik des Codes nicht ändert. Benutzen Sie in Ihrem Code jeweils nur die von der Aufgabe vorgegebene Art von Kontrollstruktur.

(a) [2 Punkte] Formen Sie die folgende for-each-Schleife in eine for-Schleife um:

Vorgabe — Java

```java
double[] values = {0.1, 0.3, 0.2, 0.3};
int index = 0;
for(double value: values) {
    System.out.print(index + " " + value);
    index++;
}
```

Ihre Lösung — Java

```java
double[] values = {0.1, 0.3, 0.2, 0.3};
```

(b) [3 Punkte] Formen Sie die folgende if-Verzweigung in eine switch-Verzweigung um:

Vorgabe — Java

```java
int choice = 2;
if(choice == 1 || choice == 3) {
    System.out.print("Wasser");
} else if(choice == 2) {
    System.out.print("Erde");
} else {
    System.out.print("ungültig");
}
```

Ihre Lösung — Java

```java
int choice = 2;
```

Aufgabe 3

_____ / 4 Punkte

(a) [3 Punkte] David möchte ein Programm schreiben, um eine große Anzahl Kundendatensätze zu verwalten. Eine bestimmte Sortierung der Daten braucht er nicht, aber in seinem Programm muss häufig geprüft werden, ob ein bestimmter Kundendatensatz vorhanden ist. David schlägt vor, die Datensätze dafür in einer `java.util.LinkedList<T>` zu speichern. Charlie sagt, dass er lieber ein `java.util.HashSet<T>` verwenden sollte.

i) Würden Sie eher Charlies oder Davids Vorschlag umsetzen? Geben Sie einen nachvollziehbaren Grund in 1–3 ausformulierten Sätzen an.

David erwidert, dass er die LinkedList bevorzugt, weil er gar nicht weiß, welche Methoden `java.util.HashSet` hat.

ii) Halten Sie dieses Argument von David für ein nachvollziehbares/starkes Argument? Begründen Sie Ihre Antwort in 1–2 ausformulierten Sätzen.

(b) [1 Punkt] Gegeben sei der reguläre Ausdruck `[a-z][a-z0-9]*`. Kreuzen Sie alle Strings an, die vollständig von diesem Ausdruck gematcht werden:

- ☐ `a123`
- ☐ `A123`
- ☐ `123a`
- ☐ `ab8p`
- ☐ `a1`
- ☐ `a`

Aufgabe 4

_____ / 10 Punkte

Für ein Schließfachsystem wird ein Programm benötigt, das prüft, ob eine Zahlenfolge gleich dem Beginn der Fibonacci-Folge ist. Die Fibonacci-Folge beginnt mit den Zahlen 0 und 1; jede weitere Zahl ist gleich der Summe ihrer beiden Vorgänger:

$0, 1, 1, 2, 3, 5, 8, \ldots$

Schreiben Sie ein Java-Programm `Fibonacci`, das beliebig viele Integer als Konsolenargumente entgegennimmt und folgende Ausgaben macht:

- `gültig`, wenn die Argumente dem Anfang der Fibonacci-Folge entsprechen
- `ungültig`, falls dies **nicht** der Fall ist oder **etwas anderes** als Zahlen übergeben wurde

Zur Erinnerung: Die Methoden zum Parsen werfen im Fehlerfall eine `NumberFormatException`.

Beispiele:

```
% java Fibonacci
gültig
% java Fibonacci 0
gültig
% java Fibonacci 0 1 1 2 3 5 8 13
gültig
% java Fibonacci 0 1 9
ungültig
% java Fibonacci 0 1 1 2 drei
ungültig
```

Fibonacci.java Java

Fibonacci.java (Fortsetzung) Java

Aufgabe 5

_____ / 13 Punkte

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

Eine natürliche Zahl n ist defizient, wenn die Summe all ihrer natürlichen Teiler **kleiner** als $2n$ ist. Ein Beispiel für eine defiziente Zahl ist 4 (Teilersumme $1 + 2 + 4 = 7 < 2 \cdot 4$).

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

(a) [4 Punkte] Schreiben Sie eine Methode `int teilersumme(int n)`, welche die Summe **aller** natürlichen Teiler von `n` berechnet.

(b) [3 Punkte] Schreiben Sie eine Methode `boolean istDefizient(int n)`, die genau dann `true` zurückgibt, wenn `n` eine defiziente Zahl ist.

(c) [6 Punkte] Schreiben Sie eine Methode `int[] defizienteZahlen(int anzahl)`, die ein Array zurückgibt, das genau die ersten `anzahl` defizienten Zahlen enthält. Falls `anzahl < 0` ist, soll eine `IllegalArgumentException` geworfen werden.

Beispiel:

```
% java DefizienteZahlen
7
true
false
1 2 3 4 5 7
```

DefizienteZahlen.java Java

```java
public class DefizienteZahlen {

    public static void main(String[] args) {
        System.out.println(teilersumme(4));
        System.out.println(istDefizient(4));
        System.out.println(istDefizient(12));

        for(int zahl: defizienteZahlen(6)) {
            System.out.print(zahl + " ");
        }
    }

    // Ergänzen Sie hier die Methoden teilersumme, istDefizient und defizienteZahlen.
```

DefizienteZahlen.java (Fortsetzung) Java

```
}
```

Aufgabe 6

_____ / 7 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;
    }
}
```

Gegeben sei die folgende Klasse:

Monster.java — Java

```
public class Monster {
    private final double weight;

    public Monster(double weight) {
        this.weight = weight;
    }

    // gibt das Gewicht des Monsters zurück
    public double weight() {
        return weight;
    }
```

Ergänzen Sie die Klasse um eine **öffentliche, statische** Methode `sorted`, die ein Array von `Monster`-Objekten übergeben bekommt, **absteigend** nach ihrem Gewicht sortiert und diese als neues `Monster`-Array zurückgibt; die Reihenfolge der Objekte im übergebenen Array soll dabei von der Methode **nicht** verändert werden. Gehen Sie davon aus, dass das übergebene Array nicht `null` ist und kein Wert im Array `null` ist.

Monster.java (Fortsetzung) — Java

Monster.java (Fortsetzung) Java

```
}
```

Aufgabe 7

_____ / 9 Punkte

Gegeben sei die folgende Klasse `StringList`, die eine einfach verkettete Liste implementiert, in der Strings gespeichert werden können:

StringList.java — Java

```java
public class StringList {
    private class Node {
        private String data;
        private Node next;

        private Node(String data, Node next) {
            this.data = data;
            this.next = next;
        }
    }

    private Node head;

    // fügt value vorne in die Liste ein
    public void prepend(String value) {
        head = new Node(value, head);
    }
```

(a) [6 Punkte] Vervollständigen Sie die Implementierung der Methode `int removeAll(String needle)`, die alle Strings aus der Liste entfernt, die den gleichen Inhalt wie `needle` haben. Der Rückgabewert soll angeben, wie viele Strings entfernt worden sind.

StringList.java (Fortsetzung) — Java

```java
  public int removeAll(String needle) {

    _____ = 0;

    while(head != null && _____ ) {

      _____;

      _____;

    }

    Node current = head;

    while(_____) {

      while(current.next != null

            && _____) {

        _____;

        current.next = _____;

      }

      _____;

    }

    _____;
  }
}
```

(b) [3 Punkte] Basierend auf der `StringList` soll eine Klasse `List` erstellt werden, die beliebige Objekte eines gemeinsamen Typs speichern kann. Bei der Erstellung einer `List` soll angegeben werden, welchen Typ die Objekte in der Liste haben (z. B. `new List<String>()`). Welche Änderungen müsste man am vorgegebenen Code der `StringList` vornehmen, um diese Anforderungen umzusetzen? *Sie können die Änderungen in Stichpunkten beschreiben, konkreten Code angeben oder die Änderungen lesbar in den vorgegeben Code eintragen.*

Aufgabe 8

_____ / 6 Punkte

Gegeben sei die Klasse `SearchTree` für einen binären Suchbaum, in dem Doubles gespeichert werden können:

SearchTree.java — Java

```java
public class SearchTree {

    private class BinaryNode {
        private double element;
        private BinaryNode left, right;

        private BinaryNode(double element) {
            this.element = element;
        }
    }

    private BinaryNode root;

    // fügt newNumber in den Baum ein
    public void insert(double newNumber) {
        // ... (Implementierung nicht abgedruckt)

    }
```

Vervollständigen Sie die Methode `int countLessThan(double t)`, die die **Anzahl** aller Einträge im Baum zurückgibt, die **kleiner** als `t` sind; bei einem leeren Suchbaum ist diese Anzahl gleich 0. Nutzen Sie in Ihrem Code die Eigenschaften eines binären Suchbaumes aus, um die Anzahl der betrachteten Knoten minimal zu halten. Sie dürfen zusätzliche Hilfsmethoden mit minimaler Sichtbarkeit schreiben.

SearchTree.java (Fortsetzung) — Java

```java
public int countLessThan(double t) {
```

SearchTree.java (Fortsetzung) Java

```
    }
}
```

Aufgabe 9

_____ / 29 Punkte

In dieser Aufgabe sollen Sie Interfaces, Klassen und Methoden für eine Marketinganwendung einer Eisdiele implementieren.

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.
- Das genaue Format von Textausgaben ist Ihnen überlassen.
- Innerhalb dieser Aufgabe müssen Sie keine Exceptions abfangen. Sie müssen keine Parameter validieren.
- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.
- Gehen Sie davon aus, dass alle in dieser Aufgabe genannten Klassen und Interfaces im selben Package liegen.

(a) [2 Punkte]
Schreiben Sie eine **öffentliches** Interface `Weather`, das eine Methode `getTemperature` ohne Parameter vorschreibt; diese Methode soll später die aktuelle Temperatur in °C zurückgeben.

Weather.java Java

(b) [5 Punkte]

Schreiben Sie eine **nicht abstrakte, öffentliche** Klasse `RandomWeather`, die das `Weather`-Interface implementiert. Als aktuelle Temperatur soll zufällig eine der Temperaturen 19 °C, 26 °C und 31°C zurückgeben werden (jeweils mit beliebiger, aber positiver Wahrscheinlichkeit).

Zur Erinnerung: Die Methode `Math.random()` gibt eine Zufallszahl z mit $0 \leq z < 1$ zurück.

RandomWeather.java Java

(c) [4 Punkte]
Erstellen Sie eine nicht abstrakte, **öffentliche** Klasse `Customer`. Jedes `Customer`-Objekt speichert eine E-Mail-Adresse und einen Namen, die beide dem **öffentlichen** Konstruktor als Parameter übergeben werden. Die E-Mailadresse und der Name sollen außerdem von **öffentliche** Methoden `getMail()` bzw. `getName()` zurückgegeben werden.

Customer.java Java

```
public class Customer {









}
```

(d) [2 Punkte]
Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `Mailer` mit einer **öffentlichen Objektmethode** `sendMail`, die keinen Rückgabewert hat und Mails wie folgt „verschickt“: Die Methode bekommt ein `Customer`-Objekt übergeben und gibt den Namen des Customers auf der Standardausgabe aus.

Mailer.java Java

```
public class Mailer {









}
```

(e) [10 Punkte] Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `Marketing`. Der **öffentliche** Konstruktor der Klasse bekommt eine `Weather`-Instanz, ein Array von `Customer`-Objekten und eine `Mailer`-Instanz übergeben und speichert diese ab.

Die Klasse soll eine **öffentliche** Methode `doMarketing` ohne Rückgabewert haben, die sich wie folgt verhält: Ist die aktuelle Temperatur (wie von dem `Weather`-Objekt angegeben) ...

- unter 20 °C, tut sie nichts.
- größer oder gleich 20 °C und kleiner als 30 °C, wird an die Hälfte (beliebig ausgewählt und gerundet) der Customers mithilfe des gespeicherten Mailers eine Mail geschickt.
- größer oder gleich 30 °C, wird an alle Customers eine Mail geschickt.

Marketing.java Java

```
public class Marketing {









}
```

(f) [6 Punkte]

Ergänzen Sie die `main`-Methode der Klasse `Gelato`, sodass folgendes passiert:

1. Eine Instanz von `RandomWeather` wird erstellt.
2. Eine Instanz von `Mailer` wird erstellt.
3. Zwei `Customer`-Objekte mit folgenden Namen und Mailadressen werden erstellt:
   - Kim, k@hhu.de
   - Alex, a@hhu.de
4. Eine Instanz von `Marketing`, der alle oben genannten Objekte in geeigneter Weise übergeben werden, wird erstellt.
5. `doMarketing` wird aufgerufen.

Gelato.java Java

```
public class Gelato {
    public static void main(String[] args) {









    }
}
```