Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Prof. Dr. Michael Schöttner

21. Juli 2023

Drittklausur

Programmierung
WS 22/23

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 17. 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	Σ
Punktzahl	11	5	8	2	25	6	6	27	90
Erreicht									

Aufgabe 1

_____ / 11 Punkte

(a) [6 Punkte] Gegeben seien die folgenden Klassen:

Studi.java

```java
public class Studi {
    private final int nr;
    private final string name;

    @Override
    public static String toString() {
        return "Matrikelnummer: " + this.nr + ", Name: " + this.name;
    }

    public Studi(int nr, String name) {
        this.nr = nr;
        this.name = name;
    }

}
```

App.java

```java
public class App {
    public static void main(String[] args) {
        // Studi mit Namen "Alex" erstellen
        Studi alex = new Studi(1234, "Alex");
        // String-Repräsentation des gerade erstellen Objekts ausgeben
        System.out.print(Studi.alex);
    }
}
```

Beim Compilieren der Klassen gibt es folgende Fehlermeldungen:

```
% javac App.java
./Studi.java:3: error: cannot find symbol
    private final string name;
                  ^
  symbol:   class string
  location: class Studi
App.java:6: error: cannot find symbol
        System.out.print(Studi.alex);
                               ^
  symbol:   variable alex
  location: class Studi
./Studi.java:6: error: toString() in Studi cannot override toString() in Object
    public static String toString() {
                         ^
  overriding method is static
./Studi.java:5: error: static methods cannot be annotated with @Override
    @Override
    ^
./Studi.java:7: error: non-static variable this cannot be referenced from a static
  context
        return "Matrikelnummer: " + this.nr + ", Name: " + this.name;
                                    ^
./Studi.java:7: error: non-static variable this cannot be referenced from a static
  context
        return "Matrikelnummer: " + this.nr + ", Name: " + this.name;
                                                           ^
6 errors
```

Geben Sie an, in welchen Klassen und 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.

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

(b) [2 Punkte] Ihr Terminal sieht gerade wie folgt aus:

```
/home/dennis % ls
projekt
/home/dennis %
```

Im Ordner `projekt` befindet sich die Datei `Go.java`. Geben Sie eine Befehlsfolge an, mit der Sie die Klasse `Go` kompilieren können.

(c) [1½ Punkte] Gegeben sei das bereits compilierte Programm `Calc`:

```
% ls
Calc.class Calc.java
```

Calc.java

```java
import java.util.Scanner;

public class Calc {
    public static void main(String[] args) {
        int result = 0;
        for(String arg: args) {
            int value = Integer.parseInt(arg);
            result += value;
        }
        System.out.print(result);
    }
}
```

Geben Sie einen Aufruf des Programms `Calc` an, sodass es `2` (und sonst nichts) auf der Standardausgabe ausgibt wird.

(d) [1½ Punkte] ChatGPT behauptet, dass es einen Compilerfehler gäbe, wenn die Zeile `import java.util.Scanner;` im Programm `Calc` entfernt wird. Stimmen Sie dieser Einschätzung zu? Begründen Sie Ihre Antwort in 1–2 ausformulierten Sätzen.

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 while-Schleife in eine for-Schleife um:

Vorgabe — Java

```java
String[] names = {"David", "Yolei", "Cody"};
int position = 1;
while(position <= names.length) {
    System.out.print(position + ": " + names[position - 1]);
    position++;
}
```

Ihre Lösung — Java

```java
String[] names = {"David", "Yolei", "Cody"};
```

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

Vorgabe — Java

```java
int choice = 3;
int cp;
if(choice == 1 || choice == 3) {
    cp = 10;
} else if(choice == 2) {
    cp = 5;
} else {
    cp = 0;
}
System.out.print(cp);
```

Ihre Lösung — Java

```java
int choice = 3;
int cp;




System.out.print(cp);
```

Aufgabe 3

_____ / 8 Punkte

Sascha möchte das Einmaleins üben. Schreiben Sie für Sascha ein Java-Programm `Mul`, das zwei ganze Zahlen a und b (in dieser Reihenfolge) als Konsolenargumente entgegennimmt und die Werte von $1 \cdot a$ bis $b \cdot a$ auf der Standardausgabe ausgibt. Die Ausgabe soll die Form `5 * 6 = 30` haben (siehe Beispiel). Falls nicht **genau zwei** oder **keine ganzen Zahlen** übergeben werden, soll sich das Programm mit der Ausgabe `err` beenden.

Zur Erinnerung: Die Parse-Methoden werfen im Fehlerfall eine `NumberFormatException`.

Beispiel-Aufrufe:

```
% java Mul -5 4
1 * -5 = -5
2 * -5 = -10
3 * -5 = -15
4 * -5 = -20
% java Mul -5 -5
% java Mul 4 -5
% java Mul 4
err
% java Mul 4 5 6
err
% java Mul vier fünf
err
```

Mul.java Java

Mul.java (Fortsetzung) Java

Aufgabe 4

_____ / 2 Punkte

(a) [1 Punkt] Kreuzen Sie alle Strings an, die vollständig vom regulären Ausdruck `[1-9][1-9][1-9][a-z][a-z][a-z][1-9][1-9][1-9]` gematcht werden:

- ☐ `111Abc444`
- ☐ `5`
- ☐ `452zoa521`
- ☐ `45acd845`
- ☐ `333xxx333`
- ☐ `a`

(b) [1 Punkt] Der oben angegebene Ausdruck ist gleichbedeutend mit:

- ☐ `[1-9][1-9][1-9][A-Z][A-Z][A-Z][1-9][1-9][1-9]`
- ☐ `[1-9][1-9]+[a-z][a-z]+[1-9][1-9]+`
- ☐ keinem dieser Ausdrücke

Aufgabe 5

_____ / 25 Punkte

Gegeben sei die folgende Klasse `Vokabel`, die eine Vokabel mit deutscher und englischer Bedeutung repräsentiert:

Vokabel.java — Java

```
public class Vokabel {
    private final String englisch;
    private final String deutsch;

    public Vokabel(String englisch, String deutsch) {
        if(englisch == null || deutsch == null) {
            throw new IllegalArgumentException();
        }
        this.englisch = englisch;
        this.deutsch = deutsch;
    }

    public String toString() {
        return englisch + ": " + deutsch;
    }

    public String getDeutsch() {
        return deutsch;
    }

    public String getEnglisch() {
        return englisch;
    }
```

(a) [6 Punkte] Schreiben Sie eine **öffentliche Klassenmethode** `boolean enthaelt(Vokabel[], String)`, die ein Array von `Vokabel`-Objekten sowie einen String übergeben bekommt. Die Methode gibt genau dann `true` zurück, wenn das Array ein Vokabel-Objekt enthält, dessen englische Bedeutung gleich dem angegebenen String ist.

Falls der Methode `null` als erster oder zweiter Parameter übergeben wird, soll eine `IllegalArgumentException` geworfen werden. Sie dürfen davon ausgehen, dass kein Wert im übergebenen Array `null` ist.

Vokabel.java (Fortsetzung) — Java

```
}
```

Ergänzen Sie die Klasse `Auflistung` um folgende **private, statische** Methoden. Sie dürfen immer davon ausgehen, dass kein Array und kein Array-Wert `null` ist.

(b) [7 Punkte] `Vokabel[] as(Vokabel[])`: Gibt ein neues Vokabel-Array zurück, das nur genau die Vokabel-Objekte aus dem übergebenen Array enthält, deren englische Bedeutung mit „a“ beginnt.
Die Klasse `String` besitzt eine öffentliche Objektmethode `boolean startsWith(String prefix)`. Diese Methode gibt genau dann `true` zurück, wenn der String, auf dem die Methode aufgerufen wird, mit dem angegeben `prefix` beginnt.

(c) [4 Punkte] `String[] enDe(Vokabel[])`: Gibt alle Vokabel-Objekte im übergebenen Array als je einen String zurück, wobei die Strings jeweils die deutsche und englische Bedeutung (in einer beliebigen Darstellungsform) enthalten.

(d) [3 Punkte] `void ausgeben(String[])`: Gibt die übergebenen Strings auf der Standardausgabe aus (lässt Reihenfolge unverändert).

(e) [5 Punkte] Vervollständigen Sie die `main`-Methode (nächste Seite), sodass folgendes passiert:

1. Ein `Vokabel`-Array, in dem die drei bereits angelegten Vokabel-Objekte referenziert werden, wird erstellt.
2. Auf der Standardausgabe wird ausgegeben, ob dieses Array eine Vokabel enthält, deren englische Bedeutung „and“ ist. (Ausgabeformat beliebig)
3. Auf der Standardausgabe werden die deutschen und englischen Bedeutungen aller Vokabeln im Array ausgegeben, deren englische Bedeutung mit „a“ beginnt.

Sie müssen dabei alle in dieser Aufgabe geschriebenen Methoden verwenden. Der Code muss auch dann korrekt funktionieren, wenn die Eigenschaften der drei vorgegebenen Vokabel-Objekte anders wären. Vergessen Sie nicht, dass `enthaelt` in einer anderen Klasse steht.

`Auflistung.java` Java

```
public class Auflistung {
```

Auflistung.java (Fortsetzung) Java

```
    public static void main(String[] args) {
        Vokabel and = new Vokabel("and", "und");
        Vokabel is = new Vokabel("is", "ist");
        Vokabel or = new Vokabel("or", "oder");

    }
}
```

Aufgabe 6

_____ / 6 Punkte

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

List.java Java

```
public class List {
    private class Node {
        private int data;
        private Node next;

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

    private Node head;

    public List(int[] initialValues) {
        for(int i = initialValues.length - 1; i >= 0; i--) {
            head = new Node(initialValues[i], head);
        }
    }

    public String toString() {
        // ... (nicht abgedruckt)

    }

    public void removeLast(int needle) {
        Node current = head;
        Node removeAfter = null;
        while(current != null) {
            if(current.next != null && current.next.data == needle) {
                removeAfter = current;
            }
            current = current.next;
        }
        if(removeAfter != null) {
            removeAfter.next = removeAfter.next.next;
        }
    }
}
```

Die Methode `void removeLast(int)` soll das letzte Vorkommen des übergebenen Integers aus der Liste entfernen. Die Methode funktioniert aber nicht richtig; beim letzten Testaufruf gibt es nicht die erwartete Ausgabe:

Test.java Java

```
public class Test {
    public static void main(String[] args) {
        int[] listElements = {-2, -5, 3, 4, 3, -2};
        List list = new List(listElements);
        System.out.println(list); // erwartet: -2, -5, 3, 4, 3, -2,
        list.removeLast(3);
        System.out.println(list); // erwartet: -2, -5, 3, 4, -2,
        list.removeLast(-2);
        System.out.println(list); // erwartet: -2, -5, 3, 4,
        list.removeLast(-2);
        System.out.println(list); // erwartet: -5, 3, 4,
    }
}
```

```
% java Test
-2, -5, 3, 4, 3, -2,
-2, -5, 3, 4, -2,
-2, -5, 3, 4,
-2, -5, 3, 4,
```

(a) [1 Punkt] Geben Sie allgemein an, in welchem Fall/in welchen Fällen die Methode nicht richtig funktioniert:

_____

_____

_____

(b) [5 Punkte] Geben Sie eine korrigierte Implementierung der Methode an; falls Sie den vorgegebenen Code nur ergänzen wollen, können Sie **eindeutig** angeben, wo Ihre Code-Ergänzungen eingefügt werden müssen.

Ihre Lösung Java

```
public void removeLast(int needle) {









}
```

Aufgabe 7

_____ / 6 Punkte

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

BinaryTree.java — Java

```
1  public class BinaryTree {
2
3      private class BinaryNode {
4          private int element;
5          private BinaryNode left, right;
6
7          private BinaryNode(int element) {
8              this.element = element;
9          }
10     }
11
12     private BinaryNode root;
13
14     public void insert(int newNumber) {
15         // ... (Implementierung nicht abgedruckt)

42     }
```

(a) [5 Punkte] Vervollständigen Sie die Methode `int count3()`, die die **Anzahl** aller Einträge im Baum zurückgibt, die restlos durch 3 teilbar sind. Sie dürfen zusätzliche Hilfsmethoden mit minimaler Sichtbarkeit schreiben.

BinaryTree.java (Fortsetzung) — Java

```
public int count3() {
```

BinaryTree.java (Fortsetzung) Java

```
    }
}
```

(b) [1 Punkt] Ist es möglich, die Eigenschaften eines binären Suchbaumes so auszunutzen, dass die Methode `count3` nicht alle Knoten betrachten muss? Begründen Sie Ihre Antwort kurz.

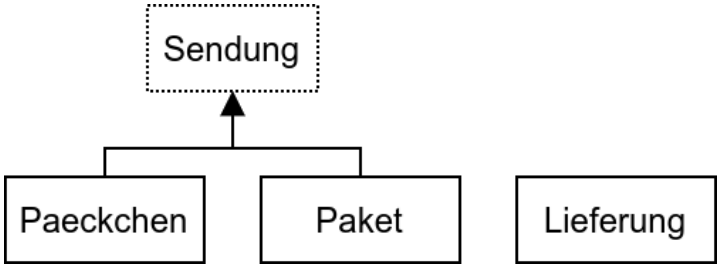
Aufgabe 8

_____ / 27 Punkte

In dieser Aufgabe sollen Sie Klassen und Methoden für ein Unternehmen schreiben, das Päckchen und Pakete innerhalb Deutschlands verschickt.

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 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) [6 Punkte]

Schreiben Sie eine **öffentliche, abstrakte** Klasse `Sendung`. Jede Sendung speichert die Postleitzahl von Absendeort und Empfängerort. Diese beiden Postleitzahlen werden über den **öffentlichen** Konstruktor gesetzt.

Schreiben Sie eine **öffentliche** Instanzmethode `boolean istKurzstrecke()`, die genau dann `true` zurückgibt, wenn Absender- und Empfängerpostleitzahl gleich sind.

Schreiben Sie eine **öffentliche, abstrakte** Methode `volumen()`, die das Volumen der Sendung zurückgibt.

Sendung.java Java

(b) [6 Punkte]

Schreiben Sie zwei nicht abstrakte, **öffentliche** Klassen `Paeckchen` und `Paket`, die sinnvoll von `Sendung` erben. Der **öffentliche** Konstruktor nimmt jeweils Absender- und Empfänger-Postleitzahl entgegen und speichert diese mithilfe der Oberklasse.

Jedes Päckchen hat ein Volumen von $3000\,\mathrm{cm}^2$; jedes Paket hat ein Volumen von $15000\,\mathrm{cm}^2$.

`Paeckchen.java` Java

`Paket.java` Java

(c) [12 Punkte]
Vervollständigen Sie die Klasse `Lieferung`, die eine Lieferung bestehend aus einer oder mehreren Sendungen repräsentiert. Ein `Lieferung`-Objekt speichert dazu ein Array von `Sendung`-Instanzen. Es gibt **zwei öffentliche** Konstruktoren, über die die Sendungen der Lieferung gesetzt werden:

- `Lieferung(Sendung[])` nimmt ein Array von Sendungen entgegen.
- `Lieferung(Sendung)` nimmt eine einzelne Sendung entgegen.

Schreiben Sie eine **private** Instanzmethode `gesamtvolumen()`, die das Gesamt-Volumen aller Sendungen der Lieferung berechnet. Schreiben Sie eine **private** Instanzmethode `boolean nurKurzstrecke()`, die genau dann `true` zurückgibt, wenn Absender- und Empfängerpostleitzahl **aller** Sendungen der Lieferung gleich sind.
Schreiben Sie eine **öffentliche** Instanzmethode `int porto()`, die das Gesamt-Porto der Lieferung berechnet. Dieses Porto ergibt sich aus dem Gesamtvolumen v nach der Formel $50 \cdot \ln(v)$. Ein Rabatt von 100 wird abgezogen, wenn Absender- und Empfängerpostleitzahl aller Sendungen der Lieferung gleich sind. Das Porto wird auf eine ganze Zahl gerundet; wie gerundet wird, dürfen Sie frei wählen.
Hinweis: `Math.log(x)` berechnet $\ln(x)$; der Rückgabewert dieser Methode ist vom Typ `double`.

Lieferung.java Java

```
public class Lieferung {
```

Lieferung.java (Fortsetzung) Java

```
}
```

(d) [3 Punkte]

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

1. Ein `Paket`-Objekt mit Absender-Postleitzahl 40215 und Empfänger-Postleitzahl 41564 wird erzeugt.
2. Eine `Lieferung`-Objekt, das nur genau dieses Paket enthält, wird erzeugt.
3. Das Gesamt-Porto dieser Lieferung wird mithilfe des `Lieferung`-Objekts berechnet und auf der Standardausgabe ausgegeben.

Versandzentrale.java Java

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









    }
}
```