Hinweise zur Probeklausur

- Wir empfehlen, die Probeklausur unter so realistischen Bedingungen wie möglich zu schreiben. Das heißt, Sie sollten sich 120 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.
    - Insbesondere werden die letzten beiden Seiten (doppelseitig) auf ein Blatt gedruckt. In dieser Probeklausur und in manchen echten Klausuren enthält dieses letzte Blatt Codevorgaben für manche Aufgaben. Dieses letzte Blatt können Sie zum einfacheren Bearbeiten der zugehörigen Aufgaben heraustrennen. Wenn eine leere Rückseite vorhanden ist, können Sie diese auch als Schmierblatt benutzen.
- Die Probeklausur wird in der letzten Woche im Tutorium besprochen. 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. Insbesondere werden die echten Klausuren auch in kleinen Aufgaben die Themen der letzten beiden Vorlesungswochen aufgreifen (siehe Selbsttests zu den Wochen 14 und 15 im Ilias).
    - Folgende Themen kommen beispielsweise in dieser Probeklausur nicht vor, sind aber klausurrelevant (unvollständige Aufzählung): Überladen, Suche, Sortieren, Klassenvariablen, abstrakte Klassen, Löschen in verketteten Listen, Generics, Fangen von Exception
    - Welche Themen klausurrelevant sind, wird in der Vorlesung kommuniziert. Insbesondere kennen die Hilfskräfte (auch diejenigen, die das Tutorium zur Vorbereitung auf die zweite Klausur halten) die Klausuren im Vorfeld nicht und können daher keine verbindlichen Aussagen zur Klausurrelevanz treffen.

- Auch in den echten Klausuren wird die letzte Aufgabe aus dem Themenbereich der objektorientierten Programmierung stammen und am meisten Punkte geben.
- Sie können die Probeklausur nutzen, um herauszufinden, mit welchen Aufgabentypen Sie besonders gut zurechtkommen; diese Aufgaben wären ein guter Startpunkt in der echten Klausur. Wenn Sie mit bestimmten Aufgabentypen Schwierigkeiten haben, versuchen Sie diese zu identifizieren und Unklarheiten zu beseitigen; überspringen Sie erstmal Aufgaben, bei denen Sie hängen.
- 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
Januar 2025

- 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!

zur Bewertung abgegebene, lose Blätter: _____ **Viel Erfolg!**

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	Σ
Punkte	9	4	12	7	13	7	12	26	90
Erreicht									

Aufgabe 1

_____ / 9 Punkte

__/5 (a) Gegeben seien die folgenden Klassen:

Person.java — Java

```java
public class Person {
    private String name;

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


    public String toString()
        return name;
    }
}
```

Studi.java — Java

```java
public class Studi extends Person {
    private int nummer;

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

    @Override
    public String toStirng() {
        return super.toString() + ", " + nummer;
    }
}
```

Beim Compilieren der Klassen gibt es folgende Fehlermeldungen:

```
./Person.java:9: error: ';' expected
    public String toString()
                           ^
./Person.java:12: error: class, interface, or enum expected
}
^
./Person.java:9: error: missing method body, or declare abstract
    public String toString()
                  ^
Studi.java:5: error: constructor Person in class Person cannot be applied to given
  types;
        super();
        ^
  required: String
  found: no arguments
  reason: actual and formal argument lists differ in length
Studi.java:9: error: method does not override or implement a method from a
  supertype
    @Override
    ^
5 errors
```

Geben Sie an, in welchen Klassen und Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehlerursachen 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; falls eine Zeile entfernt werden muss, tragen Sie `-leer-` als Korrektur ein. Geben Sie keine Folgefehler an, die durch Korrektur eines vorherigen Fehlers behoben werden.

Klasse, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Klasse, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Klasse, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Klasse, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Klasse, Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

___/2 (b) Im Terminal ist die Ordnerstruktur im aktuellen Verzeichnis dargestellt:

```
/home/jan/projects % tree
.
|-- blatt01
    |-- Hello.java
/home/jan/projects %
```

Geben Sie die Befehle an, um die Klasse `Hello` im Ordner `/home/jan/projects/blatt01` zu kompilieren und die enthaltene `main`-Methode auszuführen. *Geben Sie nur die für das Kompilieren und Ausführen notwendigen Befehle an.*

___/2 (c) Wir haben die Noten einer Klausur in der Textdatei `noten` gespeichert. Außerdem haben wir ein bereits kompiliertes Java-Programm `Schnitt`, das den Durchschnitt von Zahlen berechnen und ausgeben kann.

```
% ls
noten Schnitt.class Schnitt.java
% cat noten
1,7 1,3 1,7 2,3 2,3 1 4 3 5 3,7
```

Schnitt.java — Java

```java
import java.util.Scanner;
import java.util.LinkedList;
import java.util.List;

public class Schnitt {
    public static void main(String[] args) {
        Scanner eingabe = new Scanner(System.in);
        List<Double> zahlen = new LinkedList<>();

        // liest Zahlen ein
        while(eingabe.hasNext()) {
            double zahl = eingabe.nextDouble();
            zahlen.add(zahl);
        }

        // berechnet den Durchschnitt (den Code müssen Sie nicht nochvollziehen)
        double schnitt = zahlen.stream()
            .mapToDouble(Double::doubleValue)
            .average()
            .orElse(Double.NaN);

        // gibt Durchschnitt aus
        System.out.println(schnitt);
    }
}
```

Geben Sie einen Aufruf des Programms `Schnitt` an, der den Durchschnitt der Zahlen in der Datei `noten` berechnet und das Ergebnis in die Datei `schnitt` schreibt.

Aufgabe 2

_____ / 4 Punkte

___/2 (a) Gegeben sei folgende Klasse:

Bestellung.java Java

```java
class Bestellung {
    public void bestellung(String name, String adresse, List<Produkt> produkte) {
        // Verarbeitung der Bestellung
        // ... (Code nicht abgedruckt)
    }
}
```

Welche Aussagen über diese Klasse sind richtig?

- ☐ Die Klasse hat genau einen Konstruktor.
- ☐ Die Klasse hat genau zwei Konstruktoren.
- ☐ Es gibt einen Konstruktor, der keine Parameter hat.
- ☐ Es gibt einen Konstruktor mit drei Parametern.
- ☐ Der nicht abgedruckte Code muss mindestens ein `return`-Statement beinhalten.
- ☐ Der nicht abgedruckte Code muss mindestens ein `print`-Statement beinhalten.

___/2 (b) Für das Organisationsteam einer Vorlesung wurde ein Java-Programm entwickelt werden, das anhand einer eingelesenen Punktetabelle ausgibt, welche Studis die Zulassung erreicht haben und wie viele Punkte im Durchschnitt und Median erlangt wurden. Dazu liest das Programm die Tabellenzeilen über die Standardeingabe ein und speichert die eingelesenen Zahlen in einer Datenstruktur. Wie viele Zeilen die Tabelle hat, ist vorab nicht bekannt.

Welche Datenstruktur würden Sie zum Speichern der Tabellenzeilen benutzen? Begründen Sie Ihre Antwort in 1–2 ausformulieren Sätzen.

Aufgabe 3

_____ / 12 Punkte

In der Compact Disc Database (CDDB) werden Informationen über Audio-CDs gespeichert. CDs werden dabei über einen sogenannten Hashwert (fast eindeutig) identifiziert, der sich aus den Längen der Lieder auf der Audio-CD berechnet.

Wir betrachten als Beispiel eine CD mit 3 Liedern, die die Längen 300, 240 und 200 (in Sekunden) haben. Die Anfangszeiten der Lieder (gemessen ab Anfang der CD) ergeben sich aus den Liedlängen und sind 0, 300 und 540.

Der Hashwert ist $16^6 \cdot p_1 + 16^2 \cdot p_2 + p_3$, wobei p_1, p_2 und p_3 wie folgt berechnet werden:

- p_1: Summe der Anfangszeiten aller Lieder bestimmen, Quersumme bilden und Ergebnis modulo 255 nehmen
  im Beispiel: $0 + 300 + (300 + 240) = 840$, Quersumme 12, Modulo 255 bleibt 12
- p_2: Gesamtlänge aller Lieder: $300 + 240 + 200 = 740$
- p_3: Anzahl der Lieder: 3

Der Hashwert der Beispiel-CD ist dann $16^6 \cdot 12 + 16^2 \cdot 740 + 3 = 201516035$.

Zur Erinnerung:

- `Math.pow(x, y)` berechnet x^y und gibt einen `double` zurück.
- `123 / 10 == 12`, `123 % 10 == 3`

___/3 (a) Schreiben Sie eine **private Klassenmethode** `int digitSum(int)`, die die Quersumme der übergebenen Zahl berechnet. Die Quersumme ist die Summe der Ziffern der Zahl. Sie dürfen davon ausgehen, dass die übergebene Zahl positiv ist.

___/4 (b) Schreiben Sie eine **private Klassenmethode** `int p1(int[])`, die die Längen aller Lieder einer CD übergeben bekommt und p_1 wie oben beschrieben berechnet und zurückgibt. Gehen Sie davon aus, dass das übergebene Array nicht `null` ist.

___/2 (c) Schreiben Sie eine **private Klassenmethode** `int p2(int[])`, die die Längen aller Lieder einer CD übergeben bekommt und p_2 berechnet und zurückgibt. Gehen Sie davon aus, dass das übergebene Array nicht `null` ist.

___/3 (d) Schreiben Sie eine **öffentliche Klassenmethode** `int cddbId(int[])`, die die Längen aller Lieder einer CD übergeben bekommt und den CDDB-Hashwert berechnet und zurückgibt. Falls das übergebene Array `null` ist, soll eine `IllegalArgumentException` geworfen werden.

Cddb.java — Java

```
public class Cddb {
```

Cddb.java (Fortsetzung) Java

Cddb.java (Fortsetzung) Java

```
}
```

Aufgabe 4

_____ / 7 Punkte

___/2 (a) Die n-te sogenannte Partialsumme der alternierenden harmonischen Reihe ist definiert als:

$$f(n) = \underbrace{1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} \ldots}_{n \text{ Terme (bis einschl. } 1/n)} = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$$

(Ignorieren Sie den letzten Teil mit dem Σ, wenn Sie das Zeichen nicht kennen.)
Die folgende Methode `alternierendHarmonisch` soll $f(n)$ in Java umsetzen. Sie funktioniert allerdings nicht korrekt:

AlternierendHarmonisch.java — Java

```java
public class AlternierendHarmonisch {
    private static double alternierendHarmonisch(int n) {
        double ergebnis = 0;
        for(int k = 1; k < n; k++) {
            ergebnis += Math.pow(-1, k+1) / k;
        }
        return ergebnis;
    }

    public static void main(String[] args) {
        System.out.println(alternierendHarmonisch(1)); // erwartet: 1
        System.out.println(alternierendHarmonisch(2)); // erwartet: 0.5
        System.out.println(alternierendHarmonisch(3)); // erwartet: 0.8333
        System.out.println(alternierendHarmonisch(4)); // erwartet: 0.5833
    }
}
```

```
% java AlternierendHarmonisch
0.0
1.0
0.5
0.8333333333333333
```

Erklären Sie in 1–3 ausformulieren Sätzen, warum die Methode nicht wie gewünscht funktioniert und wie sie repariert werden kann. Zur Verdeutlichung können Sie auch in den vorgegebenen Code schreiben.

__/3 (b) Geben Sie für die beiden folgenden Codeausschnitte jeweils an, ob die idiomatische Kontrollstruktur verwendet wird. Falls die Kontrollstruktur nicht idiomatisch ist, geben Sie idiomatischen Code mit gleicher Semantik an.

Vorgabe — Java

```
public static void printChars(String string) {
    for(char c: string.toCharArray()) {
        System.out.println(c);
    }
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
public static void printChars(String string) {









}
```

Vorgabe — Java

```
public static void printReversed(String[] strings) {
    int index = strings.length;
    while(index > 0) {
        index--;
        System.out.println(strings[index]);
    }
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
public static void printReversed(String[] strings) {









}
```

___/2 (c) Geben Sie an, was das folgende Programm ausgibt, wenn es mit `java Foreach Hallo Welt` gestartet wird. Begründen Sie Ihre Antwort, indem Sie erklären, was die erste Schleife macht.

Foreach.java Java

```
public class Foreach {
    public static void main(String[] args) {
        for(String arg: args) {
            arg = arg.toUpperCase();
        }
        for(String arg: args) {
            System.out.println(arg);
        }
    }
}
```

Aufgabe 5

_____ / 13 Punkte

Zum Lernen des Alphabets haben wir uns folgendes Spiel ausgedacht: Personen sagen der Reihen nach Wörter. Der Anfangsbuchstabe des nächsten Worts muss immer weiter hinten im Alphabet liegen als der Anfangsbuchstabe des vorhergehenden Wortes. Ä, Ö und Ü zählen als A, O bzw. U. Beispiele:

- gültig: Banane, Müsli, Öl, Soja
- ungültig: Chamäleon, Kuh, Zug, Affe (A nicht weiter hinten als Z)
- ungültig: Eisbär, Ente (E nicht weiter hinten als E)

Hinweise:

- `'A' < 'Z' == true`
- `(int)'A' == 65`, `(int)'Ä' == 169`, `(int)'Ö' == 214`, `(int)'Ü' == 220`
- `String` besitzt die Objektmethoden `charAt(int)` und `length()`.
- `"abcb".replace("b", "x")` gibt den neuen String `"axcx"` zurück.

Vervollständigen Sie die Klasse `Game` durch Hinzufügen der folgenden Methoden:

___/4 (a) Eine private Klassenmethode `boolean validWord(String)`, die genau dann `true` zurückgibt, wenn das Wort mindestens 1 Zeichen lang ist und mit einem Großuchstaben von A bis Z, Ä, Ö oder Ü beginnt. Sie dürfen davon ausgehen, dass der Parameter nicht `null` ist.

___/3 (b) Eine private Klassenmethode `boolean isBefore(String, String)`, die genau dann `true` zurückgibt, wenn der zweite übergebene String auf den ersten übergegebenen String folgen darf (gemäß der Spielregeln). Sie dürfen hier davon ausgehen, dass beide Strings mit einem Großuchstaben von A bis Z, Ä, Ö oder Ü beginnen und aus jeweils mindestens einem Zeichen bestehen.

___/6 (c) Eine main-Methode, die beliebig viele Wörter als Argumente erwartet. Wenn die Wortfolge gemäß der Spielregeln gültig ist, soll `gültig` auf der Standardausgabe ausgegeben werden, ansonsten `ungültig`.[1]

Beispiele:

```
% java Game "" "A"
ungültig
% java Game "Aa" "Cc"
gültig
% java Game Aa Äa
ungültig
```

Game.java — Java

```
public class Game {
```

[1] Da alle Wörter in einer leeren Wortfolge die Spielregeln einhalten, ist eine leere Wortfolge auch gültig.

Game.java (Fortsetzung) Java

```
}
```

Aufgabe 6

_____ / 7 Punkte

Gegeben sei die folgende Klasse `List`, die eine einfach verkettete Liste implementiert, in der Integer gespeichert werden können und welche die folgenden fertigen Methoden hat:

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 = null;

    public void add(int element) {
        // fügt element am Ende der Liste ein
        // ... (Code nicht abgedruckt)

    }

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

    }
```

Außerdem gibt es folgendes Interface mit zwei fertigen Implementierungen:

Calculation.java — Java

```
public interface Calculation {
    int calc(int x);
}
```

Add2.java — Java

```
public class Add2 implements Calculation {
    public int calc(int x) {
        return x + 2;
    }
}
```

Cubed.java — Java

```
public class Cubed implements Calculation {
    public int calc(int x) {
        return x * x * x;
    }
}
```

Für die Klasse `List` soll eine **öffentliche Instanzmethode** `List map(Calculation c)` implementiert werden, die eine **neue** Liste zurückgibt, die genauso viele Elemente beinhaltet wie die aktuelle Liste. Allerdings soll statt einem Element `x` das Ergebnis von `c.calc(x)` in der neuen Liste gespeichert werden. Die alte Liste soll nicht verändert werden. Folgendes Beispiel zeigt, wie `map` funktionieren soll:

Beispiel — Java

```
List list = new List();
list.add(1);
list.add(3);
list.add(-2);
System.out.println(list); // 1,3,-2

List list2 = list.map(new Add2());
System.out.println(list2); // 3,5,0

List list3 = list.map(new Cubed());
System.out.println(list3); // Aufgabenteil a)
```

___/1 (a) Welche Zahlen gibt `System.out.println(list3)` in obigem Beispiel aus?

___/6 (b) Implementieren Sie `map`:

List.java (Fortsetzung) — Java

```
}
```

Aufgabe 7

_____ / 12 Punkte

Ein Trie ist eine baumartige Datenstruktur. Anders als bei einem binären Suchbaum kann ein Knoten (Node) allerdings eine beliebige Anzahl von Nachfolgeknoten haben.

In einen Trie wird ein Wort eingefügt, indem die Knoten – beginnend nach dem Wurzelknoten – die einzelnen Buchstaben des Wortes speichern und der letzte Knoten eine `ende`-Markierung erhält.

Es ist bereits Code fertig, den Sie **auf dem letzten Blatt** der Klausur finden. Dieses Blatt dürfen Sie abtrennen und den enthaltenen, fertigen Code in dieser Aufgabe verwenden. Auf dem Blatt finden Sie auch eine Abbildung mit einem Beispiel-Trie.

___/4 (a) Schreiben Sie eine **öffentliche Instanzmethode** `int size()`, die zurückgibt, wie viele Wörter im Trie gespeichert sind (d. h. wie viele Knoten mit `ende == true` existieren). Im angegebenen Beispiel-Trie sind 5 Wörter gespeichert. Wenn Sie Hilfsmethoden schreiben, müssen diese minimale Sichtbarkeit haben.

___/8 (b) Schreiben Sie eine **öffentliche Instanzmethode** `boolean contains(String)`, die genau dann `true` zurückgibt, wenn der übergebene String im Trie gespeichert ist. Wenn der übergebene String die Länge 0 hat oder der Methoden-Parameter `null` ist, soll eine `IllegalArgumentException` geworfen werden.

Beispiele zur Verwendung der Methoden finden Sie auf dem letzten Blatt.

Zur Erinnerung: Die Klasse `String` besitzt öffentliche Objektmethoden `char[] toCharArray()` und `int length()`.

Trie.java (Fortsetzung) Java

Trie.java (Fortsetzung) Java

```
}
```

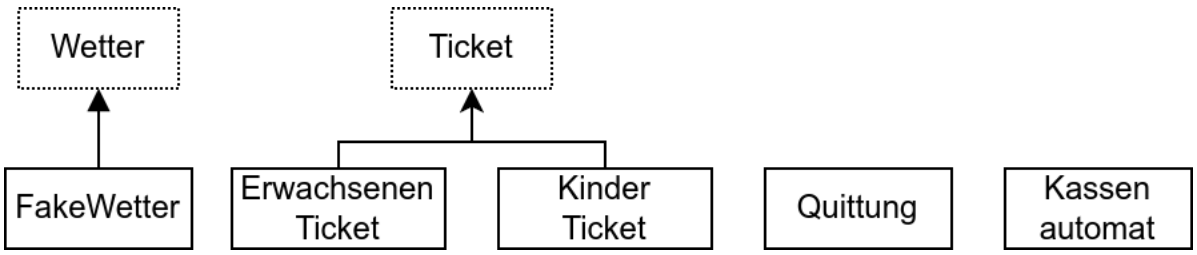
Aufgabe 8

_____ / 26 Punkte

In dieser Aufgabe sollen Sie Interfaces, Klassen und Methoden für einen Kassenautomaten eines Schwimmbads programmieren.

Hinweise:

- Sie dürfen alle Variablennamen frei wählen.
- Wählen Sie sinnvolle Datentypen für Variablen und Rückgabetypen, wenn es keine genaue Vorgabe gibt.
- Alle Instanz- und Klassenvariablen 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 nur dann Parameter validieren, wenn dies verlangt ist.
- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.



__/1 (a) Schreiben Sie ein **öffentliches** Interface `Wetter`, das eine Methode `temperatur()` ohne Parameter vorschreibt; diese Methode soll später die aktuelle Temperatur als reelle Zahl zurückgeben.

Wetter.java Java

___/1 (b) Schreiben Sie ein **öffentliches** Interface `Ticket`, das eine Methode `int preis()` ohne Parameter vorschreibt, die später den Ticketpreis in Cent zurückgibt.

Ticket.java Java

___/3 (c) Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `FakeWetter`, die das `Wetter`-Interface sinnvoll implementiert. Der Konstruktor bekommt eine Temperatur übergeben. Die `temperatur`-Methode gibt immer diese Temperatur zurück.

FakeWetter.java Java

___/6 (d) Erstellen Sie nicht abstrakte, **öffentliche** Klassen `ErwachsenenTicket` und `KinderTicket`, die das `Ticket`-Interface sinnvoll implementieren. Jedes Erwachsenen-Ticket kostet 300 Cent, jedes Kinder-Ticket 150 Cent. Überschreiben Sie außerdem die `toString`-Methode, sodass jeweils die Art des Tickets (Erwachsene/Kinder) und der Preis zurückgegeben werden.

`ErwachsenenTicket.java` Java

`KinderTicket.java` Java

___/11 (e) Vervollständigen Sie die Klasse `Quittung`. Eine Quittung speichert eine `Wetter`-Instanz und ein Array von Tickets. Der **öffentliche Konstruktor** `Quittung(int, int, Wetter)` bekommt eine Anzahl von Erwachsenen-Tickets, eine Anzahl von Kinder-Tickets und eine `Wetter`-Instanz übergeben; er legt entsprechend viele Tickets im `Ticket`-Array ab.

Schreiben Sie eine **öffentliche** Methode `int gesamtpreis()`, die den Gesamtpreis aller gespeicherten Tickets zurückgibt. Es gibt einen Rabatt von 200 Cent, wenn der Gesamtpreis (ohne Rabatt) größer als 1000 Cent und die Temperatur (wie vom `Wetter`-Objekt angegeben) größer als 30 ist. Benutzen Sie in der Methode keine `int`-Literale mit Werten größer als 1, sondern speichern Sie die Werte als Klassenkonstanten (statische, finale Attribute).

Überschreiben Sie die `toString`-Methode, sodass die String-Repräsentationen aller Tickets und der Gesamtpreis zurückgegeben werden. **(Rückseite beachten)**

Quittung.java Java

```
public class Quittung {
```

Quittung.java (Fortsetzung) Java

```
}
```

___/4 (f) Ergänzen Sie die `main`-Methode der Klasse `Kassenautomat`. Die Methode nimmt das eingeworfene Geld (in Cent), die Anzahl der Erwachsenen- und Kinder-Tickets entgegen und soll dann folgendes tun:

1. Eine Instanz von `FakeWetter` mit Temperatur 30 wird erstellt.
2. Eine Quittung wird erstellt.
3. Wenn genug Geld eingeworfen wurde, um die Quittung zu bezahlen, wird die String-Repräsentation der Quittung ausgegeben.
4. Andernfalls wird `zu wenig Geld` ausgegeben

Kassenautomat.java Java

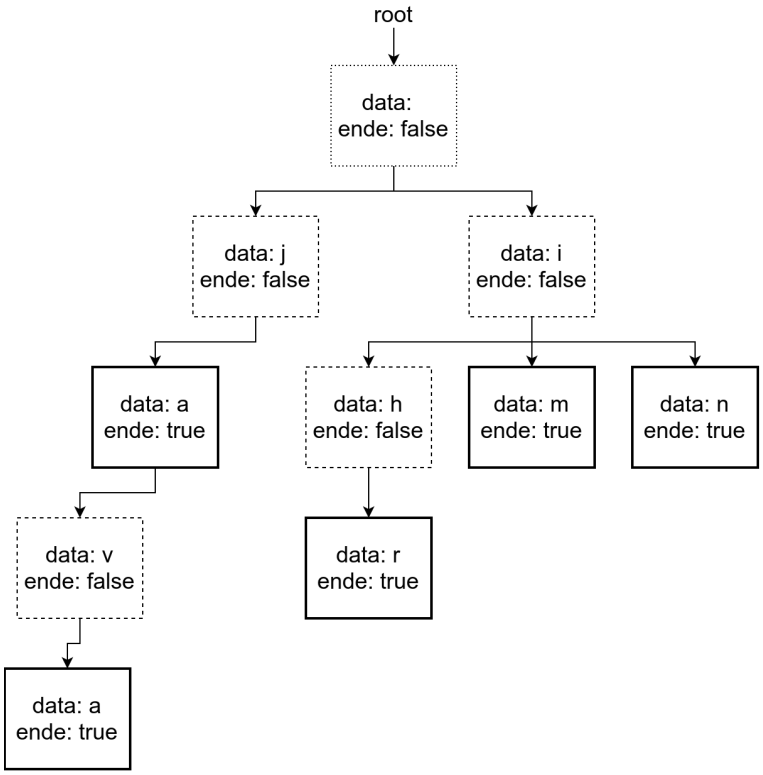
```
public class Kassenautomat {
  public static void main(String[] args) {
    int geldGegeben = Integer.parseInt(args[0]);
    int anzahlErwachsene = Integer.parseInt(args[1]);
    int anzahlKinder = Integer.parseInt(args[2]);

    Wetter w = _____;

    Quittung q = _____;


    if(_____) {

      _____;

    } else {

      _____;

    }
  }
}
```

Code zu Aufgabe 7

Trie.java Java

```
public class Trie {
    private class Node {
        private char data;
        private Node[] next; // niemals null, _nicht_ sortiert, keine Duplikate
        private boolean ende;

        private Node(char data, Node[] next, boolean ende) {
            this.data = data;
            this.next = next;
            this.ende = ende;
        }
    }

    private Node root = new Node(' ', new Node[]{}, false);

    public void add(String wort) {
        // fügt wort in den Trie ein
        // ... (Code nicht abgedruckt)

    }
```

In folgendem Beispiel-Trie sind die fünf Wörter ja, java, ihr, in und im gespeichert:



Beispielverwendung der Methoden Java

```
Trie trie = new Trie();
trie.add("java");
trie.add("ja");
trie.add("im");
trie.add("in");
trie.add("ihr");

System.out.println(trie.size()); // 5
System.out.println(trie.contains("ja")); // true
System.out.println(trie.contains("jav")); // false
System.out.println(trie.contains("java")); // true
System.out.println(trie.contains("javas")); // false
System.out.println(trie.contains(" ")); // false
System.out.println(trie.contains("")); // löst IllegalArgumentException aus
```

Sie dürfen dieses Blatt abtrennen und als Schmierzettel benutzen. Dieses Blatt muss nicht, darf aber abgegeben werden. Wenn das Blatt abgetrennt ist und bewertet werden soll, muss es als loses Blatt auf dem Deckblatt angegeben werden.