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

24. März 2023

Nachklausur

Programmierung
WS 22/23

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

Hinweise:

- Diese Klausur enthält 22 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	9	5	7	24	6	6	6	27	90
Erreicht									

Aufgabe 1

_____ / 9 Punkte

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

Person.java

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

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

    @Override
    public String toString() {
        retrun name;
    }
}
```

Studi.java

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

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

    @Override
    public String toString() {
        retrun super.toString() + " " + nr;
    }
}
```

Beim Compilieren der Klassen gibt es folgende Fehlermeldungen:

```
Studi.java:4: error: <identifier> expected
    public Studi(String name, intnr) {
                                   ^
Studi.java:6: error: not a statement
        this.nr == nr;
                ^
Studi.java:11: error: not a statement
        retrun super.toString() + " " + nr;
        ^
Studi.java:11: error: ';' expected
        retrun super.toString() + " " + nr;
              ^
Studi.java:11: error: not a statement
        retrun super.toString() + " " + nr;
                                        ^
5 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) [1 Punkt] Ihr Terminal sieht gerade wie folgt aus:

```
/home/fabian/project % ls
Hello.java
```

Mit welchem Befehl können Sie die Klasse `Hello` kompilieren? *Geben Sie nur den für das Kompilieren notwendigen Befehl an.*

(c) [2 Punkte] Bei einer Klausur mit drei Teilnehmer:innen wurden die Punktzahlen 87, 44 und 69 erreicht. Wir haben ein bereits compiliertes Java-Programm `Maximum`, das das Maximum von ganzen Zahlen berechnen und ausgeben kann.

```
% ls
punkte Maximum.class Maximum.java
% cat punkte
87 44 69
```

Maximum.java Java

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

public class Maximum {
    public static void main(String[] args) {
        List<Integer> zahlen = new LinkedList<>();

        // liest Zahlen ein
        for(String arg: args) {
            int zahl = Integer.parseInt(arg);
            zahlen.add(zahl);
        }

        // berechnet das Maximum (den Code müssen Sie nicht nochvollziehen)
        int maximum = zahlen.stream()
            .mapToInt(Integer::intValue)
            .max()
            .orElse(0);

        // gibt das Maximum aus
        System.out.println(maximum);
    }
}
```

Geben Sie einen Aufruf des Programms `Maximum` an, sodass das Maximum der Zahlen 87, 44 und 69 berechnet und auf der Standardausgabe ausgegeben wird.

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 = {"Tai", "Matt", "Sora"};
int index = 0;
while(index < names.length) {
    System.out.print(index + ": " + names[index]);
    index++;
}
```

Ihre Lösung — Java

```java
String[] names = {"Tai", "Matt", "Sora"};
```

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

Vorgabe — Java

```java
int choice = 3;
if(choice == 1 || choice == 3) {
    System.out.print("10 CP");
} else if(choice == 2) {
    System.out.print("5 CP");
} else {
    System.out.print("err");
}
System.out.print(":)");
```

Ihre Lösung — Java

```java
int choice = 3;

System.out.print(":)");
```

Aufgabe 3

_____ / 7 Punkte

Kim möchte des Einmaleins üben. Schreiben Sie für Kim ein Java-Programm `Mal`, das eine ganze Zahl x als Konsolenargument entgegennimmt und die Produkte $1 \cdot x$ bis $10 \cdot x$ als ganze Zahlen auf der Standardausgabe ausgibt. Die Ausgabe soll die Form `2 * 5 = 10` haben (siehe Beispiel). Falls **nicht genau ein** Argument oder **keine ganze Zahl** übergeben wird, soll sich das Programm mit der Ausgabe `err` beenden.

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

Beispiel-Aufrufe:

```
% java Mal 3
1 * 3 = 3
2 * 3 = 6
3 * 3 = 9
4 * 3 = 12
5 * 3 = 15
6 * 3 = 18
7 * 3 = 21
8 * 3 = 24
9 * 3 = 27
10 * 3 = 30
% java Mal zwei 3
err
% java Mal zwei
err
% java Mal
err
```

Mal.java Java

Mal.java (Fortsetzung) Java

Aufgabe 4

_____ / 24 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 = 1; 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 `Vokabel`, die eine Vokabel mit Lernstand, deutscher und englischer Bedeutung repräsentiert.

Vokabel.java

```java
public class Vokabel {
    private final String englisch;
    private final String deutsch;
    private int lernstand;

    public Vokabel(String englisch, String deutsch) {
        this.englisch = englisch;
        this.deutsch = deutsch;
        this.lernstand = 0;
    }

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

    public String getDeutsch() {
        return deutsch;
    }

    public String getEnglisch() {
        return englisch;
    }

    public void richtigBeantwortet() {
        lernstand++;
    }

    public int getLernstand() {
        return lernstand;
    }
```

(a) [6 Punkte] Vervollständigen Sie die Klassenmethode `nachDeutsch`, die ein Array von `Vokabel`-Objekten übergeben bekommt und ein neues Array zurückgeben soll, in dem dieselben Objekte **aufsteigend** nach ihrer deutschen Übersetzung (Instanzvariable `deutsch`) lexikographisch sortiert sind (A–Z); die Reihenfolge der Objekte im übergebenen Array soll dabei von der Methode **nicht** verändert werden; das Original-Array und das sortierte Array sollen dieselben Objekte im Heap referenzieren.

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

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`

Vokabel.java (Fortsetzung) Java

```
public static _____________ nachDeutsch(Vokabel[] vokabelnOrig) {
 if(vokabelnOrig _____ null) {
   ____________________________________________________________;
 }
 Vokabel[] vokabeln = new Vokabel[_____________________________];
 for(int i = 0; i < vokabeln.length; i++) {
   ____________________________________________________________;
 }
 for(int i = 1; i _____ vokabeln.length; i++) {
   ____________________________________________________________;
   int einfPos = i;
   while(einfPos > 0 &&
    vokabeln[einfPos-1].deutsch.compareTo(_______.deutsch) __ 0)){
     ____________________________________________________________;
     einfPos--;
   }
   ____________________________________________________________;
 }
 ____________________________________________________________;
}
}
```

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[] nichtGelernt(Vokabel[])`: Gibt ein neues Vokabel-Array zurück, das nur genau die Vokabel-Objekte aus dem übergebenem Array enthält, deren Lernstand 0 ist.

(c) [4 Punkte] `String[] deutsch(Vokabel[])`: Gibt die deutschen Bedeutungen aller Vokabel-Objekte im übergebenen Array zurück (lässt Reihenfolge unverändert)

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

(e) [4 Punkte] Vervollständigen Sie die `main`-Methode (nächste Seite), sodass die deutschen Bedeutungen aller bereits angelegten Vokabeln, die den Lernstand 0 haben, alphabetisch sortiert ausgegeben werden. Sie müssen dabei alle in dieser Aufgabe geschriebenen Methoden verwenden; vergessen Sie nicht, dass `nachDeutsch` in einer anderen Klasse steht. Der Code muss auch dann korrekt funktionieren, wenn die Eigenschaften der drei vorgegebenen Vokabel-Objekte anders wären.

`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");

        or.richtigBeantwortet();

    }
}
```

Aufgabe 5

_____ / 6 Punkte

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

List.java 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 removeFirst(int needle) {
        if(head != null && head.data == needle) {
            head = head.next;
            return;
        }
        while(head != null) {
            if(head.next != null && head.next.data == needle) {
                head.next = head.next.next;
                return;
            }
            head = head.next;
        }
    }

}
```

Die Methode `void removeFirst(int)` soll das erste 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

```java
public class Test {
    public static void main(String[] args) {
        int[] listElements = {2, 7, 1, 8, 2, 8};
        List list = new List(listElements);
        System.out.println(list); // erwartet: 2, 7, 1, 8, 2, 8,
        list.removeFirst(2);
        System.out.println(list); // erwartet: 7, 1, 8, 2, 8,
        list.removeFirst(8);
        System.out.println(list); // erwartet: 7, 1, 2, 8,
    }
}
```

```
% java Test
2, 7, 1, 8, 2, 8,
7, 1, 8, 2, 8,
1, 2, 8,
```

Geben Sie eine korrigierte Implementierung der Methode an:

Ihre Lösung Java

```
public void removeFirst(int needle) {




}
```

Aufgabe 6

_____ / 6 Punkte

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

Tree.java — Java

```java
public class Tree {

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

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

    private BinaryNode root;

    public void insert(double newNumber) {
        // ... (Implementierung nicht abgedruckt)

    }
```

Vervollständigen Sie die Methode `int countNegative()`, die die **Anzahl** aller Einträge im Baum zurückgibt, die **kleiner** als 0 sind; bei einem leeren Suchbaum ist die 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.

Tree.java (Fortsetzung) — Java

```java
public int countNegative() {
```

Tree.java (Fortsetzung) Java

```
    }
}
```

Aufgabe 7

_____ / 6 Punkte

(a) [1 Punkt] Gegeben sei der reguläre Ausdruck `[RGB][0-9]+`. Kreuzen Sie alle Strings an, die vollständig von diesem Ausdruck gematcht werden:

- ☐ `r2`
- ☐ `3G`
- ☐ `G4`
- ☐ `R1`
- ☐ `B12`
- ☐ `G`

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

- ☐ `[RGB][0-9][0-9]*`
- ☐ `[RGB][0-9][0-9]`
- ☐ `[RGB]+[0-9]+`

(c) [1½ Punkte] Wir wollen in einer Datenstruktur speichern, wer und in welcher Reihenfolge zuletzt unsere Mikrowolle benutzt hat. Zac schlägt vor, ein Hashset zu benutzen. Vane meint, es sollte besser eine einfach verkette Liste verwendet werden.

Würden Sie eher Zacs oder Vanes Vorschlag umsetzen? Geben Sie einen nachvollziehbaren Grund in 1–2 ausformulierten Sätzen an.

(d) [2½ Punkte] Im folgenden Programm soll eine verkettete Liste zum Speichern von Strings verwendet werden. Es gibt die fertige JDK-Klasse `java.util.LinkedList<E>`. Vervollständigen Sie den folgenden Code, sodass er funktioniert:

Demo.java — Java

```
_____ _____.LinkedList;

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

        _____ namen = new LinkedList<>();
        namen.add("Ashley");
        System.out.println(namen.get(0));

    }
}
```

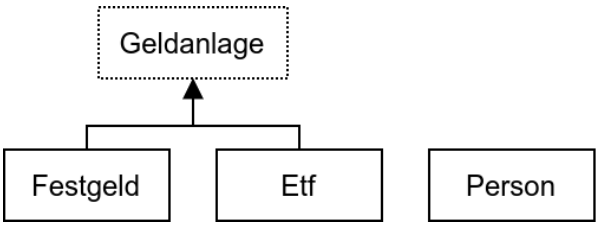
Aufgabe 8

_____ / 27 Punkte

In dieser Aufgabe sollen Sie Klassen und Methoden zur Vermögensberechnung programmieren.

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) [6 Punkte]

Schreibe Sie eine **öffentliche, abstrakte** Klasse `Geldanlage`, die ein Startkapital (in Cent, ganze Zahl) und einen Zinssatz (rationale Zahl) abspeichert. Diese beiden Informationen werden dem **öffentlichen** Konstruktor übergeben und werden von **öffentlichen** Methoden `startkapital()` und `zinssatz()` zurückgegeben.

Schreiben Sie eine **abstrakte, öffentliche** Methode `int geldNach(int jahre)`, die später berechnen soll, auf welches Endkapital das Startkapital nach der angegebenen Anzahl von Jahren voraussichtlich angewachsen ist.

Geldanlage.java Java

(b) [5 Punkte]

Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `Festgeld`, die sinnvoll von `Geldanlage` erbt. Der **öffentliche** Konstruktor nimmt Startkapital und Zinssatz entgegen und speichert sie mithilfe der Oberklasse ab. Das Endkapital nach n Jahren bei einem Startkapital K_0 und einem Zinssatz p ergibt sich durch $K_0 \cdot (1 + p \cdot n)$. Beachten Sie, dass das Endkapital eine ganze Zahl ist; wie gerundet wird, dürfen Sie entscheiden.

Festgeld.java Java

(c) [5 Punkte]

Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `Etf`, die sinnvoll von `Geldanlage` erbt. Der **öffentliche** Konstruktor nimmt ein Startkapital entgegen und speichert es mithilfe der Oberklasse ab; der Zinssatz ist immer 0,07. Das Endkapital nach n Jahren bei einem Startkapital K_0 und einem Zinssatz p ergibt sich durch $K_0 \cdot (1 + p)^n$. Beachten Sie, dass das Endkapital eine ganze Zahl ist; wie gerundet wird, dürfen Sie entscheiden.

Zur Erinnerung: `Math.pow(a, b)` berechnet a^b und gibt einen **Double** zurück.

`Etf.java` `Java`

(d) [7 Punkte]

Vervollständigen Sie die Klasse `Person`, die eine Person mit verschiedenen Geldanlagen repräsentiert. Dem **öffentlichen** Konstruktor wird ein Array von `Geldanlage`-Instanzen und ein Name übergeben.

Schreiben Sie eine **öffentliche** Methode `int geldNach(int jahre)`, die die voraussichtliche Summe der Endkapitale aller Geldanlagen der Person nach der angegebenen Anzahl von Jahren zurückgibt. Überschreiben Sie die Methode `toString`, sodass der Name und die voraussichtliche Summe der Endkapitale nach 5 Jahren zurückgegeben werden. **(Rückseite beachten)**

Person.java Java

```
public class Person {









}
```

(e) [4 Punkte]

Ergänzen Sie die folgende `main`-Methode, sodass folgendes passiert:

1. Eine Festgeld-Instanz mit 10000 Cent Startkapital und einem Zinssatz von 0,01 wird erstellt.
2. Eine Etf-Instanz mit 10000 Cent Startkapital wird erstellt.
3. Eine Person namens Sascha mit den beiden zuvor erstellten Geldanlagen wird erstellt.
4. Die String-Repräsentation der Person wird auf der Standardausgabe ausgegeben.

Test.java Java

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









    }
}
```