HHU Düsseldorf
Institut für Informatik
Dr. Markus Brenneis

Daten des Prüflings

Matrikelnummer: _____

Nachname: _____

Vorname: _____

3. Klausur

Programmierung WS 23/24
02. Mai 2024

- Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- 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	6	7	8	Σ
Punkte	7	6	7	23	9	7	4	27	90
Erreicht									

Aufgabe 1

_____ / 7 Punkte

___/1 (a) Ihr Terminal sieht gerade wie folgt aus:

```
% ls
Jumble.java
```

Geben Sie einen Befehl an, mit dem die Datei `Jumble.java` kompiliert werden kann (sodass also die Datei `Jumble.class` entsteht). Geben Sie nur den fürs Kompilieren notwendigen Befehl an.

___/6 (b) Der Code der Klasse sieht wie folgt aus:
Hinweis: Im JDK gibt es in der Klasse `java.util.Arrays` die öffentliche, statische Methode `void sort(char[])`.

Jumble.java — Java

```java
import java.util.Arrays.sort;

public class Jumble {
    public void printTask(String word) {
        char[] letters = word.toCharArray();
        Arrays.sort(letters);
        for(char[] letter: letters) {
            System.out.print(letter);
        }
        return System.out.println(": ");
    }
}
```

Beim Kompilieren der Klassen gibt es folgende Fehlermeldungen:

```
Jumble.java:1: error: cannot find symbol
import java.util.Arrays.sort;
                        ^
  symbol:   class sort
  location: class Arrays
Jumble.java:6: error: cannot find symbol
        Arrays.sort(letters);
        ^
  symbol:   variable Arrays
  location: class Jumble
Jumble.java:7: error: incompatible types: char cannot be converted to char[]
        for(char[] letter: letters) {
                           ^
Jumble.java:10: error: incompatible types: unexpected return value
        return System.out.println(": ");
                                  ^
4 errors
```

Geben Sie an, in welchen Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehlerursache 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, und keine neuen Fehler entstehen. 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: _____

Aufgabe 2

_____ / 6 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.

___/3 (a) Formen Sie die folgende while-Schleife in eine do-while-Schleife um.

Vorgabe Java

```
System.out.println("Start");
double r = Math.random();
while(r > 0.1) {
    System.out.print(r);
    r = Math.random();
}
System.out.print(r);
System.out.println("End (nur die letzte Ausgabe ist <= 0.1)");
```

Ihre Lösung Java

```
System.out.println("Start");









System.out.println("End (nur die letzte Ausgabe ist <= 0.1)");
```

___/3 (b) Formen Sie die folgende if-Verzweigung in eine switch-Verzweigung um.

Vorgabe Java

```
char num = '2';
String output;
if(num == '0') {
    output = "no";
} else if(num == '1') {
    output = "a";
} else {
    output = "many";
}
System.out.println(output);
```

Ihre Lösung Java

```
char num = '2';
String output;









System.out.println(output);
```

Aufgabe 3

_____ / 7 Punkte

Zur Analyse von DNA-Strings wollen wir wissen, wie oft ein Buchstabe in einem String gleich dem vorherigen Buchstaben ist.

Vervollständigen Sie das Programm `Repeated`, das genau einen String als Argument erwartet und ausgibt, wie oft ein Buchstabe im Argument gleich dem direkt davor stehenden Buchstaben ist. Falls nicht genau ein Argument übergeben wird, soll nichts ausgegeben werden.

Beispielaufrufe:

```
% java Repeated
% java Repeated ABAB
0
% java Repeated AAAB
2
% java Repeated ABAAB
1
```

Zur Erinnerung: `"abc".charAt(1)` hat den Wert *b*; `"abc".length()` hat den Wert 3.

Repeated.java Java

```
public class Repeated {









}
```

Aufgabe 4

_____ / 23 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 `Studi`, die Studis mit Namen und erreichten Übungspunkten repräsentiert.

Studi.java

```java
public class Studi {

    private final String name;
    private final int punkte;

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

    public int getPunkte() {
        return punkte;
    }

    public String getName() {
        return name;
    }
```

___/6 (a) Vervollständigen Sie die Klassenmethode `nachPunktzahl`, die ein Array von `Studi`-Objekten übergeben bekommt und ein **neues** Array zurückgeben soll, in dem dieselben Objekte **absteigend** nach ihrer Punktzahl sortiert sind; 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.

Studi.java (Fortsetzung) Java

```
  public static _______________ nachPunktzahl(Studi[] studisOrig) {
    if(studisOrig == null) {
      ____________________________________________________________;
    }
    _________________ studis = new Studi[_____________________________];
    for(int i = 0; i < studis.length; i++) {
      ____________________________________________________________;
    }
    for(int i = _______________; i < studis.length; i++) {
      ____________________________________________________________;
      int einfPos = i;
      while(einfPos > 0
       && ____________________________________________________) {
        ____________________________________________________________;
        einfPos--;
      }
      ____________________________________________________________;
    }
    ____________________________________________________;
  }
}
```

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

___/7 (b) `Studi[] zugelassen(Studi[])`: Gibt ein neues Studi-Array zurück, das nur genau die Studi-Objekte aus dem übergebenen Array enthält, deren Punktzahl **größer oder gleich** 100 ist; die Reihenfolge der Objekte soll nicht verändert werden.

___/4 (c) `String[] namen(Studi[])`: Gibt die Namen aller Studi-Objekte im übergebenen Array zurück (lässt Reihenfolge unverändert).

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

___/3 (e) Vervollständigen Sie die `main`-Methode (nächste Seite), sodass die Namen der Studis in `studis` mit einer Punktzahl ≥ 100 ausgegeben werden, wobei die Studis absteigend nach Punktzahl sortiert sind. Sie müssen dabei alle in dieser Aufgabe geschriebenen Methoden verwenden; vergessen Sie nicht, dass `nachNote` in einer anderen Klasse steht. Der Code muss auch dann korrekt funktionieren, wenn die Eigenschaften der drei vorgegebenen Objekte anders wären.

Auflistung.java Java

```
public class Auflistung {
```

Auflistung.java (Fortsetzung) Java

```
    public static void main(String[] args) {
        Studi husk = new Studi("Husk", 160);
        Studi vaggie = new Studi("Vaggie", 54);
        Studi charlie = new Studi("Charlie", 145);

        Studi[] studis = {husk, vaggie, charlie};

    }
}
```

Aufgabe 5

_____ / 9 Punkte

Eine doppelt verkettete Liste ähnelt der einfach verketteten Liste aus der Vorlesung. Es gibt aber folgende zwei Ergänzungen:

- Jeder Knoten (Node) speichert auch eine Referenz auf den Vorgänger (`previous`); beim ersten Element ist der Vorgänger `null`.
- Die Liste speichert eine Referenz `tail` auf das letzte Element der Liste.

Die folgende Klasse `LinkedList` soll eine doppelt verkettete Liste umsetzen, in der `int`-Werte gespeichert werden:

LinkedList.java — Java

```
public class LinkedList {
    private class Node {
        private int data;
        private Node next;
        private Node previous;

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

    private Node head = null;
    private Node tail = null;

    public String toString() {
        // gibt die Zahlen beginnend bei head bis einschließlich tail zurück
        // (Code nicht abgedruckt)

    }
```

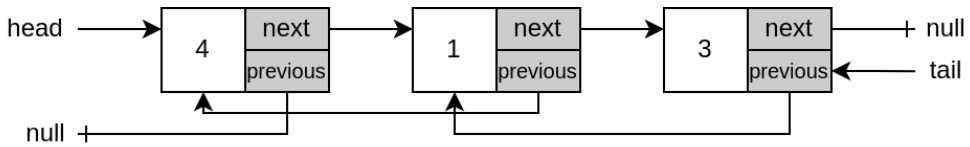
Das folgende Beispiel zeigt, wie die `LinkedList` funktionieren soll:

Beispiel — Java

```
LinkedList zahlen = new LinkedList();
zahlen.add(3);
zahlen.add(1);
zahlen.add(4);

System.out.println(zahlen); // 4, 1, 3,
System.out.println(zahlen.reversed()); // 3, 1, 4,
```

Nach diesem Beispiel sieht die `LinkedList` intern so aus:



___/5 (a) Vervollständigen Sie die Methode `void add(int)`, die die übergebene Zahl vorne in die (ggf. leere) Liste einfügt.

___/4 (b) Vervollständigen Sie die Methode `String reversed()`, die die Elemente der Liste von hinten nach vorne als String zurückgibt. Die Zahlen sollen mit Kommata voneinander getrennt sein (vgl. Beispiel oben). Die Liste darf durch die Methode nicht verändert werden.

LinkedList.java (Fortsetzung) Java

```
public void add(int element) {









}
public String reversed() {









}
```

Aufgabe 6

_____ / 7 Punkte

Gegeben sei die folgende Klasse `Tree`, die einen binären Suchbaum implementiert, in dem Integer gespeichert werden können, und die folgende fertige Methoden hat:

Tree.java

```java
public class Tree {
    private class BinaryNode {
        private int element;
        private BinaryNode left, right;

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

    private BinaryNode root;

    public void insert(int newNumber) {
        // ... (Code nicht abgedruckt)

    }

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

    }
```

Außerdem gibt es folgendes Interface mit drei fertigen Implementierungen:

Condition.java

```java
public interface Condition {
    boolean check(int x);
}
```

IsEven.java

```java
public class IsEven implements Condition {
    public boolean check(int n) {
        return n % 2 == 0;
    }
}
```

IsSmall.java

```java
public class IsSmall implements Condition {
    public boolean check(int x) {
        return x < 10;
    }
}
```

IsNegative.java

```java
public class IsNegative implements Condition {
    public boolean check(int n) {
        return n < 0;
    }
}
```

Für die Klasse `Tree` soll eine Instanzmethode `boolean allMatch(Condition c)` implementiert werden, die für jede Zahl `x` im Baum einmal `c.check(x)` ausführt. Genau dann, wenn für **alle** Zahlen im Baum die Rückgabe von `check` `true` ist, soll `allMatch` `true` zurückgeben.

Folgendes Beispiel zeigt, wie `allMatch` funktionieren soll (x sind die Zahlen im Baum):

Beispiel — Java

```
Tree tree = new Tree();
System.out.print(tree.allMatch(new IsEven())); // true (alle x erfüllen x%2 == 0)
tree.insert(1);
tree.insert(-2);
tree.insert(3);
tree.insert(-3);
System.out.print(tree); // -3,-2,1,3
System.out.print(tree.allMatch(new IsEven())); // false (nicht alle x erfüllen x%2 == 0)
System.out.print(tree.allMatch(new IsSmall())); // true (alle x erfüllen x < 10)
System.out.print(tree.allMatch(new IsNegative())); // Aufgabenteil a)
```

___/1 (a) Was ist der Wert von `tree.allMatch(new IsNegative())` in diesem Beispiel? Begründen Sie Ihre Antwort kurz.

___/6 (b) Implementieren Sie `allMatch`; Sie müssen dabei eine private, rekursive Hilfsmethode benutzen:

Tree.java (Fortsetzung) — Java

```
public boolean allMatch(Condition c) {



}
```

Aufgabe 7

_____ / 4 Punkte

___/1 (a) Kreuzen Sie alle Strings an, die vollständig vom regulären Ausdruck `[a-d]*[0-9]+[a-z]*` gematcht werden:

- ☐ `abcdef`
- ☐ `abc123def`
- ☐ `f1`
- ☐ `b1b`
- ☐ `123`

___/1 (b) Welche dieser Ausdrücke sind gleichbedeutend mit dem obigen Ausdruck?

- ☐ `[a-d][a-d]*[0-9][0-9]*[a-z][a-z]*`
- ☐ `[a-d]*[0-9][0-9]*[a-z]*`
- ☐ `[a-d]*[0-9][0-9]*[a-z]*[a-z]*`

___/2 (c) Angenommen, Sie speichern 1 Mio. Zahlen in einer Datenstruktur. Die erste Zahl (an Position 0) soll um 1 erhöht werden. Ist diese Operation in einer einfach verketteten Liste wesentlich langsamer als in einem Array? Begründen Sie Ihre Antwort mit einem Argument in 1–3 ausformulierten Sätzen.

Aufgabe 8

_____ / 27 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.

___/$3\frac{1}{2}$ (a) 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. Überschreiben Sie `toString()`, sodass alle im `Customer`-Objekte gespeicherten Informationen zurückgegeben werden.

Customer.java — Java

```
public class Customer {




}
```

___/2 (b) Schreiben Sie eine **öffentliches** Interface `Mailer`, das eine Methode `sendMail` ohne Rückgabe vorschreibt. Dieser Methode soll später ein `Customer`-Objekt übergeben werden, um dem Customer eine Mail zu schicken.

Mailer.java Java

___/$2\frac{1}{2}$ (c) Schreiben Sie eine **nicht abstrakte, öffentliche** Klasse `FakeMailer`, die das `Mailer`-Interface implementiert und Mails wie folgt „verschickt“: Die String-Repräsentation des übergebenen `Customer`-Objekts wird auf der Standardausgabe ausgegeben.

FakeMailer.java Java

___/$4\frac{1}{2}$ (d) Fügen Sie der Klasse `Weather` eine Methode `getTemperature` ohne Parameter hinzu. Die Methode soll die „aktuelle“ Temperatur in °C zurückgeben. Sie soll zufällig eine der Temperaturen $-5\,°C$, $21\,°C$ und $36\,°C$ zurückgeben (jeweils mit beliebiger, aber positiver Wahrscheinlichkeit).
Zur Erinnerung: Die Methode `Math.random()` gibt eine Zufallszahl z mit $0 \leq z < 1$ zurück.

Weather.java Java

```
public class Weather {




}
```

__/10 (e) Ergänzen Sie die 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 {









}
```

___/$4\frac{1}{2}$ (f) Ergänzen Sie die `main`-Methode der Klasse `Gelato`, sodass folgendes passiert:

1. Eine Instanz von `FakeMailer` wird erstellt.
2. Eine Instanz von `Weather` wird erstellt.
3. Eine Instanz von `Marketing`, der alle oben genannten Objekte sowie die zwei bereits angelegten `Customer`-Objekte in geeigneter Weise übergeben werden, wird erstellt.
4. `doMarketing` wird aufgerufen.

Gelato.java Java

```
public class Gelato {
    public static void main(String[] args) {
        Customer c1 = new Customer("blitzo@hhu.de", "Blitzo");
        Customer c2 = new Customer("loona@hhu.de", "Loona");









    }
}
```