HHU Düsseldorf
Institut für Informatik
Prof. Dr. Michael Schöttner

Daten des Prüflings

Matrikelnummer: _____

Nachname: _____

Vorname: _____

1. Klausur

Programmierung WS 24/25
10. Februar 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	Σ
Punkte	6	5	4	19	8	21	27	90
Erreicht								

Aufgabe 1

_____ / 6 Punkte

Gegeben sei das folgende Programm, das das Maximum der ersten zwei Argumente ausgeben soll:

Max.java — Java

```java
public class Max {
    public static void main(String[] args) {
        if(args.length =< 1) {
            System.out.print("at least 2 arguments expected");
        } else() {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            int maximum < Math.max(a, b);
            System.out.print(maximum);
        }
    }

}
```

Beim Compilieren der Klasse gibt es folgende Fehlermeldungen:

```
% javac Max.java
Max.java:3: error: illegal start of type
        if(args.length =< 1) {
                        ^
Max.java:3: error: ';' expected
        if(args.length =< 1) {
                         ^
Max.java:5: error: -> expected
        } else() {
                ^
Max.java:8: error: ';' expected
            int maximum < Math.max(a, b);
                        ^
Max.java:8: error: > or ',' expected
            int maximum < Math.max(a, b);
                                  ^
Max.java:8: error: not a statement
            int maximum < Math.max(a, b);
                                    ^
Max.java:8: error: ';' expected
            int maximum < Math.max(a, b);
                                     ^
Max.java:8: error: not a statement
            int maximum < Math.max(a, b);
                                      ^
Max.java:8: error: ';' expected
            int maximum < Math.max(a, b);
                                       ^
Max.java:5: error: not a statement
        } else() {
               ^
Max.java:10: error: ';' expected
        }
         ^
Max.java:5: error: 'else' without 'if'
        } else() {
          ^
12 errors
```

Geben Sie an, in welchen 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.

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Zeilennummer: _____

Fehlerursache: _____

korrigierte Codezeile: _____

Aufgabe 2

_____ / 5 Punkte

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 printAllWords(String[] words) {
    for(int i = 0; i < words.length; i++) {
        System.out.println(words.get(i));
    }
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
public static void printAllWords(String[] words) {

}
```

Vorgabe — Java

```
public static void printPositions(String[] names) {
    int position = 1;
    while(position <= names.length) {
        System.out.println(position + ": " + names[position - 1]);
        position++;
    }
}
```

☐ ist idiomatisch

☐ ist nicht idiomatisch; folgender Code wäre idiomatisch:

ggf. idiomatischen Code angeben — Java

```
public static void printPositions(String[] names) {

}
```

Aufgabe 3

_____ / 4 Punkte

Wir haben ein Java-Projekt mit den zwei öffentlichen Klassen `MainApp` und `Helper`. Im Terminal ist die Ordnerstruktur im aktuellen Verzeichnis dargestellt:

```
/home/kristin/projects % tree
.
|-- src
    |-- progra
        |-- util
            |-- Helper.java
        |-- MainApp.java
/home/kristin/projects %
```

Der Classpath ist `/home/kristin/projects/src`. Die Datei `MainApp.java` liegt im Ordner `/home/kristin/projects/src/progra` und benutzt – neben Klassen aus `java.lang` – nur die Klasse `Helper` im Ordner `/home/kristin/projects/src/progra/util`.

___/$1\frac{1}{2}$ (a) Geben Sie die voll-qualifizierten Namen der Klassen `MainApp` und `Helper` an:

`MainApp`: _____

`Helper`: _____

___/$2\frac{1}{2}$ (b) Geben Sie die ersten beiden Statements in der Datei `MainApp.java` an:

_____

_____

```
public class MainApp {
```

Aufgabe 4

_____ / 19 Punkte

Bei der Atbash-Verschlüsselung wird jeder Buchstabe wie folgt geändert: A wird zu Z, B wird zu Y, C wird zu X usw. Der n-te Buchstabe im Alphabet wird also durch den $27 - n$-ten Buchstaben ersetzt, also z. B. der 2. Buchstabe B durch den 25. Buchstaben Y.

Beispiel: `HALLOX` → `SZOOLC`

Aus der Vorlesung wissen wir, dass `'A' < 'Z' == true`, `(int)'A' == 65`, `'B' - 'A' == 1`, `('A' + 25) == 90` und `(char)('A' + 25) == Z`.

Zur Erinnerung: `String` besitzt die Instanzmethoden `charAt(int)`, `length()` und `toCharArray()` und den Konstruktor `String(char[])`.

__/2 (a) Kreuzen Sie alle Ausdrücke an, die den Wert 4 haben:

☑ `'D' - 64` ☑ `'E' - 'A'` ☐ `'D' - 65` ☐ `'D' - 'A'`

__/4 (b) Schreiben Sie eine **private Klassenmethode** `boolean isValidInput(String)`, die genau dann `true` zurückgibt, wenn der übergebene String nur Großbuchstaben (von `'A'` bis `'Z'`) enthält.

__/8 (c) Schreiben Sie eine **private Klassenmethode** `String atbash(String)`, die den übergebenen String mit dem Atbash-Verfahren verschlüsselt und den verschlüsselten String zurückgibt. Denken Sie an notwendige Type-Casts.

__/5 (d) Schreiben Sie eine `main`-Methode. Das erste Argument soll Atbash-verschlüsselt und der verschlüsselte String auf der Standardausgabe ausgegeben werden. Wenn **nicht genau** ein Argument übergeben wird oder das Argument **nicht nur** aus den **Buchstaben** A – Z besteht, soll sich das Programm mit der Ausgabe `err7999` beenden.

```
% java Atbash
err7999
% java Atbash abc
err7999
% java Atbash ABC
ZYX
```

A 65

Atbash.java Java

```
public class Atbash {
```

Atbash.java (Fortsetzung) Java

```
}
```

Aufgabe 5

_____ / 8 Punkte

Gegeben sei die folgende Klasse `BinaryTree`, die einen binären Suchbaum implementiert, in dem Integer gespeichert werden können. Die Klasse hat einige fertige Methoden, die Sie benutzen können:

BinaryTree.java — Java

```java
public class BinaryTree {
    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() {
        return toString(root);
    }

    private String toString(BinaryNode current) {
        if(current == null) {
            return "";
        }
        return toString(current.left) + current.element + "," + toString(current.right);
    }
```

Wie aus der Vorlesung bekannt befindet sich ganz links im Baum die kleinste gespeicherte Zahl.

Implementieren Sie für `BinaryTree` eine **öffentliche Objektmethode** `toArrayReversed()`, die die Zahlen im Baum **ab**steigend sortiert als int-Array zurückgibt; falls der Baum leer ist, soll ein leeres Array zurückgegeben werden. Folgendes Beispiel zeigt, wie `toArrayReversed` funktionieren soll:

Beispiel — Java

```java
BinaryTree tree = new BinaryTree();
tree.insert(5);
tree.insert(-2);
tree.insert(1);
tree.insert(3);
System.out.println(tree); // -2,1,3,5,

int[] reversed = tree.toArrayReversed();
System.out.println(reversed[0]); // 5
System.out.println(reversed[1]); // 3
System.out.println(reversed.length); // 4
```

Zur Erinnerung: `"1=28=-4=".split("=")` gibt ein String-Array mit den drei Elementen `"1"`, `"28"` und `"-4"` zurück. `"".split("=")` gibt ein String-Array mit genau einem Element, dem leeren String, zurück.

BinaryTree.java (Fortsetzung) Java

```
}
```

Aufgabe 6

_____ / 21 Punkte

In einer Bibliothek gibt es eine Liste von Büchern. Jedes Buch speichert, ob es bereits ausgeliehen ist oder nicht. In der Bibliothek dürfen nicht mehr als 50 % aller Bücher ausgeliehen sein.

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.

Vervollständigen Sie wie folgt die vorgegebene Implementierung.

___/$4\frac{1}{2}$ (a) Schreiben Sie für `LinkedList` einen Konstruktor, der eine andere `LinkedList`-Instanz `other` übergeben bekommt, und alle Elemente aus `other` zu der gerade konstruierten, noch leeren Liste hinzufügt.

Verwendungsbeispiel — Java

```java
LinkedList<String> l1 = new LinkedList<>();
l1.add("a");
l1.add("b");
LinkedList<String> l2 = new LinkedList<>(l1);
// l1 und l2 enthalten jetzt beide "a", "b" (in dieser Reihenfolge).
```

LinkedList.java — Java

```java
public LinkedList(LinkedList<T> other) {









}
```

___/$8\frac{1}{2}$ (b) Implementieren Sie die Methode `ausleihen(String titel)` in der Klasse `Bibliothek`. Die Methode sucht in der Liste `buecher` (Zeile 2 in `Bibliothek`) nach **einem** Buch, das den angegebenen Titel hat und ausleihbar ist. Beachten Sie, dass es ggf. mehrere Bücher mit demselben Titel geben kann.

Die Methode soll genau dann `false` zurückgeben, wenn kein Buch mit dem angegebenen Titel ausleihbar ist; ein Buch ist **nicht** ausleihbar, wenn

- bereits mehr als 50 % aller Bücher der Bibliothek ausgeliehen sind, oder
- das Buch bereits ausgeliehen ist.

Wenn ein Buch mit dem Titel ausleihbar ist, dann muss

- das Buch-Objekt auf ausgeliehen gesetzt werden und
- die Anzahl der ausgeliehenen Bücher in der Bibliothek aktualisiert werden.

Bibliothek.java Java

```
public boolean ausleihen(String titel) {




}
```

___/3 (c) Tatsächlich müssen wir in unserer Bibliothek häufig nachsehen, ob ein Buch mit einem bestimmten Titel vorhanden und ausleihbar ist. Ist eine einfach verkettete Liste dafür eigentlich die sinnvollste Datenstruktur? Begründen Sie Ihre Antwort und machen Sie ggf. einen Vorschlag für eine bessere Datenstruktur.

___/5 (d) Gehen Sie für diese Teilaufgabe davon aus, dass Aufgabenteil (b) korrekt gelöst ist und keine weiteren Methoden zur `Bibliothek` hinzugefügt werden. Die Methode `ausleihen` in der Klasse `Bibliothek` soll ja eigentlich sicherstellen, dass nicht mehr als 50 % der Bücher ausgeliehen werden können. Betrachten wir folgenden Code, der außerhalb der Klasse `Bibliothek` liegt:

Codebeispiel — Java

```
1 Bibliothek x = new Bibliothek();
2 x.buchHinzufuegen("A");
3 x.buchHinzufuegen("A");
4 x.buchHinzufuegen("B");
5 x.ausleihen("A");
6 x.ausleihen("A");
7 LinkedList<Buch> bs = x.alleBuecher();
```

Geben Sie den Rückgabewert des Aufrufs in Zeile 6 an: _____

Ist es möglich, diesen Java-Code so zu erweitern, dass er dafür sorgt, dass in der Liste `buecher` der Instanz `x` mehr als die Hälfte der Bücher ausgeliehen sind?

☐ Nein, das ist nicht möglich. ☐ Ja, das ist möglich.

Begründen Sie Ihre **angekreuzte** Antwort, indem Sie auf den **Heap** und **Referenzen** eingehen und nachvollziehbar (z. B. mit konkretem Code) angeben, **wie** der Zustand, bei dem zu viele Bücher ausgeliehen sind, erreicht werden kann, bzw. erklären, warum dies nicht möglich ist.

Aufgabe 7

_____ / 27 Punkte

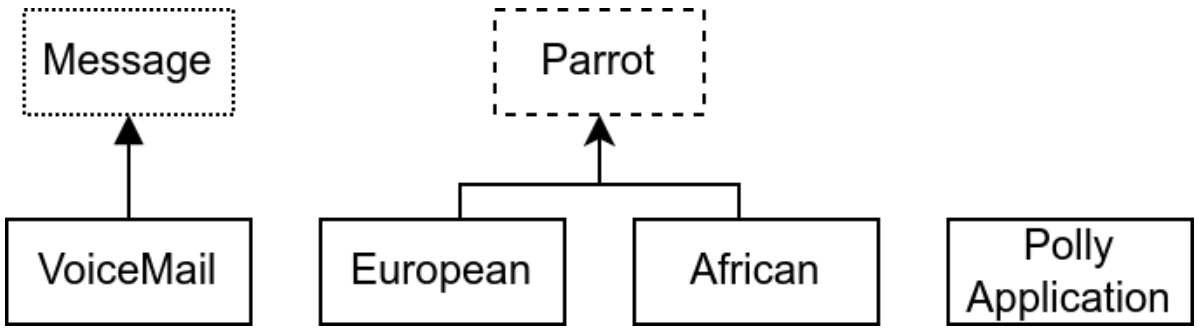
Wir haben eine revolutionäre Methode entwickelt, um Nachrichten zu verschicken: Wir geben Papageien (Parrots) eine Nachricht vor, diese fliegen zum Empfänger und der Papagei spricht die Nachricht vor.

Es gibt verschiedene Arten von Papageien und Nachrichten.

In dieser Aufgabe sollen Sie Klassen und Methoden für unsere Parrot-Firma 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.
- Gehen Sie davon aus, dass alle in dieser Aufgabe genannten Klassen und Interfaces im selben Package liegen.



Bereits fertig ist das Interface `Message`:

Message.java — Java

```
public interface Message {
    String speak(); // gibt vor, wie diese Nachricht vorgelesen wird
}
```

___/$4\frac{1}{2}$

(a) Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `VoiceMail`, die das `Message`-Interface sinnvoll implementiert. Der Konstruktor bekommt einen String übergeben und speichert diesen ab. Wenn die Voicemail vorgelesen wird, soll genau dieser String vorgelesen werden.

VoiceMail.java — Java

___/$5\frac{1}{2}$ (b) Erstellen Sie eine **abstrakte, öffentliche** Klasse `Parrot`. Der **Konstruktor** bekommt ein Objekt vom Typ `Message` übergeben und speichert es ab.

Jeder `Parrot` hat eine öffentliche Methode `void read()`, die zuerst `"Nachricht lautet: "` auf der Standardausgabe ausgibt und danach den von `Message.speak()` vorgegebenen String.

Die Parrots haben unterschiedliche Fluggeschwindigkeiten. Schreiben Sie eine öffentliche, **abstrakte** Methode `double getSpeed()`.

Parrot.java Java

___/$10\frac{1}{2}$ (c) Schreiben Sie (auf der nächsten Seite) zwei nicht abstrakte, **öffentliche** Klassen `European` und `African`, die sinnvoll von `Parrot` erben und zwei Papageien-Arten darstellen. Die **Konstruktoren** bekommen jeweils ein `Message`-Objekt übergeben und speichern es mithilfe der Oberklasse ab.

Ein `European`-Parrot hat immer eine Geschwindigkeit von 12.

Der **Konstruktor** für den `African`-Parrot bekommt zusätzlich eine Anzahl von Kokosnüssen n übergeben. Falls diese Anzahl **negativ** ist, soll im Konstruktor eine `IllegalArgumentException` geworfen werden. Die Geschwindigkeit ergibt sich aus der Formel $\max(0, 12 - 9 \cdot n)$, d. h. es wird grundsätzlich $12 - 9 \cdot n$ berechnet, das Ergebnis ist aber nie negativ.

European.java Java

African.java Java

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

1. Eine Voicemail mit dem Inhalt „Foo999“ wird erstellt.
2. Ein `African`-Parrot mit einer Kokosnuss und der gerade erstellen Voicemail wird erstellt.
3. Die Geschwindigkeit des erstellen Parrots wird auf der Standardausgabe ausgegeben.
4. Die Voicemail wird vom Parrot vorgelesen.

PollyApplication.java — Java

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









    }
}
```

___/2 (e) Angenommen, wir packen alle Klassen aus dieser Aufgabe in einer jar-Datei zusammen; in der jar-Datei befinden sich dann nur die class-Dateien der Klassen. Die jar-Datei fügen wir in den Classpath für ein neues, größeres Projekt unserer Parrot-Firma ein.

Kreuzen Sie die korrekten Aussagen an:

- ☐ Es ist möglich, in dem neuen Projekt eine weitere Implementierung `Postcard` des Interfaces `Message` zu erstellen.
- ☐ Es ist möglich, dem Konstruktor der Klasse `European` ein solches `Postcard`-Objekt zu übergeben.
- ☐ Es ist **nicht** möglich, dem Konstruktor der Klasse `European` ein solches `Postcard`-Objekt zu übergeben, weil die Klasse `Postcard` noch nicht existiert hat, als `European` kompiliert worden ist.

Klausurnr. 999 (Z99)

Code zu Aufgabe 6

LinkedList.java Java

```java
public class LinkedList<T> {
    private class Node {
        private T data;
        private Node next;

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

    private Node head = null;

    public LinkedList() {
    }

    public LinkedList(LinkedList<T> other) {
        // Aufgabenteil a)
    }

    public void add(T element) {
        // fügt element am Ende der Liste ein (fertiger Code nicht abgedruckt)
    }

    public int size() {
        // gibt Anzahl der Elemente in der Liste zurück (fertiger Code nicht abgedruckt)
    }

    public T get(int index) {
        // gibt das Element am gegebenen Index zurück (fertiger Code nicht abgedruckt)
    }
}
```

Buch.java Java

```java
public class Buch {
    private final String titel;
    private boolean ausgeliehen = false;

    public Buch(String titel) {
        this.titel = titel;
    }

    public void ausleihen() {
        if(!ausgeliehen) {
            ausgeliehen = true;
        }
    }

    public void zurueckgeben() {
        ausgeliehen = false;
    }

    public boolean ausleihbar() {
        return !ausgeliehen;
    }

    public boolean hatTitel(String titel) {
        return this.titel.equals(titel);
    }
}
```

Bibliothek.java Java

```java
public class Bibliothek {
    private final LinkedList<Buch> buecher = new LinkedList<>();
    private int ausgelieheneBuecher = 0;

    public boolean ausleihen(String titel) {
        // Aufgabenteil b)
    }

    public LinkedList<Buch> alleBuecher() {
        return new LinkedList<>(buecher);
    }

    public void buchHinzufuegen(String titel) {
        buecher.add(new Buch(titel));
    }
```

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.