Chair of Computer Graphics and Visualization
CIT
Technische Universität München

E0576
Sticker mit SRID hier einkleben

Hinweise zur Personalisierung:

- Ihre Prüfung wird bei der Anwesenheitskontrolle durch Aufkleben eines Codes personalisiert.
- Dieser enthält lediglich eine fortlaufende Nummer, welche auch auf der Anwesenheitsliste neben dem Unterschriftenfeld vermerkt ist.
- Diese wird als Pseudonym verwendet, um eine eindeutige Zuordnung Ihrer Prüfung zu ermöglichen.

Einführung in die Informatik

Klausur: IN0001 / Endterm **Datum:** Montag, 12. Februar 2024
Prüfer: Prof. Dr. R. Westermann **Uhrzeit:** 08:00 – 10:00

	A 1	A 2	A 3	A 4	A 5	A 6
I						

Bearbeitungshinweise

- Diese Klausur umfasst **24 Seiten** mit insgesamt **6 Aufgaben**.
  Bitte kontrollieren Sie jetzt, dass Sie eine vollständige Angabe erhalten haben.
- Die Gesamtpunktzahl in dieser Klausur beträgt 77 Punkte.
- Das Heraustrennen von Seiten aus der Prüfung ist untersagt.
- Als Hilfsmittel sind zugelassen:
  - ein **analoges Wörterbuch** Deutsch ↔ Muttersprache **ohne Anmerkungen**
- Schreiben Sie weder mit roter / grüner Farbe noch mit Bleistift.
- Schalten Sie alle mitgeführten elektronischen Geräte vollständig aus, verstauen Sie diese in Ihrer Tasche und verschließen Sie diese.
- Schreiben Sie ihre Lösungen immer in der gefragten Reihenfolge in die gegebenen Textfelder.
- Mögliche Exceptions und deren Behandlung werden in dieser Klausur ignoriert.

Hörsaal verlassen von _____ bis _____ / Vorzeitige Abgabe um _____

cit-eidi-1-20240212-E0576-01

Aufgabe 1 Programmverstehen (17 Punkte)

0 1 2 3 4 5 6 7

a) Gegegeben ist das folgende Java-Programm. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt werden kann. Geben Sie die Ausgaben des Programms untereinander und in der Reihenfolge in der sie generiert werden an. Tipp: Beginnen Sie mit der Bearbeitung an der main-Prozedur. Schreiben Sie die i-te Ausgabe in der Form: i) Ausgabe

```
class Element implements Comparable<Element> {

   private String name;
   private float density;

   public Element(String name, float density) {
       this.name = name;
       this.density = density;
   }

   public String toString() { return name; }

   public float getDensity() { return density; }

   public int compareTo(Element other) {

       if(this.getDensity() > other.getDensity()) {
           return -1;
       }
       else if(this.getDensity() < other.getDensity()) {
           return 1;
       }
       else {
           return 0;
       }
   }
}

class CallArray {

    static int[] arr = {24, -7, 123456, 25, -64};

    CallArray() {
        for(int i=0; i < arr.length; i++)
            arr[i] = i;
    }

    int[] arrayFunc(int[] arr) {

        for(int i=0;i < arr.length;i++)
            arr[i] = 100;

        arr = this.arr;

        return arr;
    }
}
```

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList {
    Node head;

    public LinkedList() {
        this.head = null;
    }

    public void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }

    public void doSomethingWithList() {
        Node current = head;
        while (current != null) {
            Node node = current;
            Node temp = current.next;
            while (temp != null) {
                if (temp.data > node.data) {
                    node = temp;
                }
                temp = temp.next;
            }

            int tempData = current.data;
            current.data = node.data;
            node.data = tempData;

            current = current.next;
        }
    }
}
```

```java
public class Test {

    public static int myFunc(int x, int y) {

        for (int r; (r = x % y) != 0; x = y, y = r) { }

        return y;
    }

    public static void main(String[] b) {

        int l = 0;
        int k = 5;

        for(; k > l; ) {
            ++l;
            k--;
        }

        System.out.println("k:" + k + " l:" +l);                // Ausgabe 1)


        Element[] feld = {
            new Element("Lithium", 0.5f),
            new Element("Kupfer", 8.9f),
            new Element("Aluminium", 2.7f),
            new Element("Chrom", 7.1f)
        };

        Arrays.sort(feld);

        System.out.println(feld[0] + " " + feld[1] + " " +
                feld[2] + " " + feld[3]);                       // Ausgabe 2)



        int[] arr1 = {24, -7, 123456, 25, -64};
        int[] arr2 = arr1;

        for(int i=0;i < arr1.length;i++)
            arr1[i] = 10;

        System.out.println("1: " + arr2[2]);                    // Ausgabe 3)

        CallArray arr = new CallArray();
        arr2 = arr.arrayFunc(arr1);

        System.out.println("1: " + arr1[2] + " 2: " + arr2[2]); // Ausgabe 4)


        String[][] personen = {
                { "Maximilian", "Anzinger"},
                { "Rudiger", "Westermann" },
                { "Ali", "Kocal"}};
        String[] person = new String[2];
        person[0] = "Donald";
        person[1] = "Trump";
        personen[1] = person;

        for (int i = 0; i < personen.length; i++) {
            for (int j = 0; j < personen[i].length; j++) {
                System.out.print(personen[i][j] + " ");         // Ausgabe 5)
```

```
            }
        }


        int myVal = myFunc(16,24);
        System.out.print("myVal: " + myVal);                    // Ausgabe 6)



        LinkedList list = new LinkedList();
        list.addNode(64);
        list.addNode(25);
        list.addNode(12);
        list.addNode(22);
        list.addNode(11);

        list.doSomethingWithList();

        Node temp = list.head;
        while (temp != null) {
            System.out.print(temp.data + " ");                  // Ausgabe 7)
            temp = temp.next;
        }
    }
}
```

Hier Ihre Lösungen

0
1
2

b)

Gegegeben ist das folgende Java-Programm. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt werden kann. Ergänzen Sie die Methode **switchCallMethod** an den gekennzeichneten Lücken, so dass das Programm ausgiebt: "first rest". Beachten Sie, dass in dieser Aufgabe in ihrem Code für eine Lücke kein Semikolon enthalten sein darf.

Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.

Ihren Code für die i-te Lücke schreiben Sie in der Form: i) Code.

```
enum DaysOfWeek {
      Monday,
      Tuesday,
      Wednesday,
      Thursday,
      Friday,
      Saturday,
      Sunday;

      static DaysOfWeek day1 = Monday;
      static DaysOfWeek day2 = Tuesday;
}
```

```
public class Test {

    public static String switchCallMethod(DaysOfWeek day) {

        return(switch(day) {
            ____________(1)__________;
            ____________(2)__________;
        });
    }

    public static void main(String[] b) {

        // Hier die Ausgabe des Programms
        System.out.println (
            switchCallMethod(DaysOfWeek.day1)
            + " " +
            switchCallMethod(DaysOfWeek.day2)
        );
    }
}
```

Hier Ihre Lösungen

c)

0 1 2 3 4

Gegegeben ist das folgende Java-Programm. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt werden kann. Geben Sie die Ausgaben des Programms untereinander und in der Reihenfolge in der sie generiert werden an. Tipp: Beginnen Sie mit der Bearbeitung an der main-Prozedur. Beachten Sie, dass die Ausgaben über mehrere Zeilen gehen und in den aufgerufenen Methoden gemacht werden.

Schreiben Sie die i-te Ausgabe in der Form: i) Ausgabe

```java
class Counter {
    int cnt1 = 5;
    int cnt2;

    void inc() {
        cnt1 = cnt1 + 1;
        cnt2 = cnt2 + 1;
    }

    public Counter() {
        inc();
        cnt1 = cnt2 = 0;

    }
}

class MyCounter1 extends Counter {
    int cnt1 = 5;

    void inc() {
        cnt1 = cnt1 + 2;
        cnt2 = cnt2 + 2;
        System.out.println("cnt1: " + cnt1 + " cnt2: " + cnt2);
    }

    public MyCounter1() {
        System.out.println("cnt1: " + cnt1 + " cnt2: " + cnt2);
    }
}

class MyCounter2 extends MyCounter1 {
    int cnt1;

    void inc() {
        cnt1 = cnt1 + 3;
        cnt2 = cnt2 + 3;
        System.out.println("cnt1: " + cnt1 + " cnt2: " + cnt2);
    }
}


public class Test {

    public static void main(String[] b) {

        Counter m = new MyCounter1();                    // Ausgabe 1)
        m = new MyCounter2();                            // Ausgabe 2)

    }
}
```

Hier Ihre Lösungen

d)

0 1 2 3 4

Gegegeben ist das folgende Java-Programm. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt werden kann, wenn Sie die Methode **allChargers** richtig implementieren. Die Ausgabe des Programms soll sein: ”electric charger, fuel charger”. Ergänzen Sie das Programm an den gekennzeichneten Lücken.
Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.
Ihren Code für die i-te Lücke schreiben Sie in der Form: i) Code.

```
class Car {

    interface Loading {
        public void chargerType();
    }

    public void allChargers() {

        class ElectricCar implements Loading {

            public void chargerType() {
                System.out.print("electric charger, ");
            }
        }

        Loading conventionalCar = ______(1)_______ {


            ______(2)_______ {
                ______(3)_______;
            }
        __(4)__

        (new ElectricCar()).chargerType();
        conventionalCar.chargerType();
    }
}


public class Test {

    public static void main(String[] b) {

        Car cars = new Car();
        cars.allChargers();
    }
}
```

Hier Ihre Lösungen

cit-eidi-1-20240212-E0576-07

Aufgabe 2 Mehrgestaltigkeit (12 Punkte)

0 1 2 3 4 5

a)
Gegegeben ist das folgende Java-Programm. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt werden kann. Geben Sie die Ausgaben des Programms untereinander und in der Reihenfolge in der sie generiert werden an.
Schreiben Sie die i-te Ausgabe in der Form: i) Ausgabe

```
class Animal {

    public void func(Animal a, Animal b){
        System.out.println("A1");
    }

    public void func(Animal a, Primate b){
        System.out.println("A2");
    }
}

class Mammal extends Animal {

    public void func(Animal a, Animal b){
        System.out.println("M1");
    }

    public void func(Animal a, Mammal b){
        System.out.println("M2");
    }

    public void func(Mammal a, Mammal b){
        System.out.println("M3");
    }

    public void func(Primate a, Primate b){
        System.out.println("M4");
    }
}

class Primate extends Mammal {

    public void func(Animal a, Animal b){
        System.out.println("R1");
    }

    public void func(Animal a, Primate b){
        System.out.println("R2");
    }

    public void func(Mammal a, Primate b){
        System.out.println("R3");
    }

    public void func(Primate a, Primate b){
        System.out.println("R4");
    }
}
```

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

        Animal anim;
        Mammal mam;
        Primate prim;

        mam = new Primate();
        mam.func(new Animal(), new Animal());                    Ausgabe 1)

        prim = new Primate();
        rep.func(new Animal(), new Mammal());                    Ausgabe 2)

        anim = new Primate();
        anim.func(new Primate(), new Primate());                 Ausgabe 3)

        mam =  (Mammal) new Reptile();
        mam.func(new Primate(), new Primate());                  Ausgabe 4)

        prim =  new Primate();
        anim.func((Animal) new Mammal(), (Animal) new Mammal());  Ausgabe 5)

    }
}
```

Hier ihre ihre Lösungen.

cit-eidi-1-20240212-E0576-09

0
1
2
3

b)
Geben Sie jeweils für die Code-Segmente 1) bis 3) an, welches der Ereignisse Compiler-Fehler, Laufzeit-Fehler, fehlerfreie Programmausführung eintritt. Gehen Sie davon aus, dass jedes Code-Segment separat übersetzt wird. Geben Sie die Ereignisse untereinander und in der Reihenfolge 1) bis 3) an.
Schreiben Sie das Ereignis für den Code i in der Form: i) Ereignis

```
class CreepyHouse<E> implements Comparable<E> {

    private E creep;
    public void setCreep(E x) {
        creep = x;
    }
    public E getCreep() {
        return creep;
    }
    public int compareTo(E obj) { return 0;}
}

class Creepy{}

class Murder extends Creepy {}

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

        // 1)
        CreepyHouse<Creepy> house = new CreepyHouse<Murder>();

        // 2)
        Comparable<Creepy> comp = new CreepyHouse<Creepy>();

        // 3)
        CreepyHouse<Creepy> house = new CreepyHouse<>();
        house.setCreep(new Murder());
    }
}
```

Hier ihre Lösungen.

0
1
2

c)
Gegeben ist die teilweise Beschreibung des funtionalen Interface **Function** und einer seiner default Implementierungen. **Erklären Sie**, ob der darauffolgende Code zu einem Laufzeit-Fehler oder einem Compiler-Fehler führt, oder der Code fehlerfrei läuft. In letztem Fall geben Sie das ausgegebene Ergebnis an.

```
public interface Function<T,R> {...

    default <V> Function<T,V> andThen(Function<? super R, ? extends V> after)

}
```

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

        // berechnet das Quadrat der eingegebenen Zahl
        Function<Integer, Float> half = a -> 2.0f * a * a;

        // halbiert die eingegebene Zahl
        half = half.andThen(a -> a / 2.0);

        System.out.println(half.apply(10));
    }
}
```

Geben Sie hier ihre Lösung an.

d)
Gegeben sind die folgenden 4 Code-Zeilen, die in einer prinzipiell lauffähigen Java-Klasse verwendet werden. Die Beschreibung der Klassen Anima, Mammal und Reptile finden Sie in Aufgabe 2a. Geben Sie für jede Zeile an, ob der Code zu einem Laufzeit-Fehler oder einem Compiler-Fehler führt, oder der Code fehlerfrei läuft. Geben Sie die Ereignisse untereinander und in der Reihenfolge 1) bis 4) an.
Schreiben Sie das Ereignis für den Code i in der Form: i) Ereignis

0
1
2

```java
1) List<? extends Object> foo1 = new ArrayList<Animal>();
2) List<? extends Mammal> foo2 = new ArrayList<Reptile>();
3) List<? extends Reptile> foo3 = new ArrayList<Mammal>();
4) List<? super Animal> foo4 = new ArrayList<Object>();
```

Geben Sie hier ihre Lösungen an.

cit-eidi-1-20240212-E0576-11

Aufgabe 3 Stacks (9 Punkte)

Der Shunting-yard-Algorithmus überführt mathematische Ausdrücke bestehend aus Operanden-Zeichen, Operatoren-Zeichen und Klammern von der Infix-Notation in die Postfix-Notation.
Beispiel: (a+4)*(5-b) = a4+5b-*
Sie können davon ausgehen, dass Variablen-Bezeichner nur aus einem Buchstaben-Zeichen bestehen, nur Ganzzahlen von 0 bis 9 und nur die Operatoren + (Addition), - (Subtraktion), * (Multiplikation), / (Division) und Klammern (,) vorkommen.
Ein Eingabe-String, der den Ausdruck in Infix-Notation enthält, wird in einen Ausgabe-String konvertiert, der den Ausdruck in Postfix-Notation enthält. Der Algorithmus benötigt weiterhin einen Stack, der für das Zwischenspeichern der Operator-Zeichen benötigt wird.
Der Eingabe-String wird zeichenweise gelesen, wobei alle Operanden-Zeichen (Variablen und Zahlen) direkt und in derselben Reihenfolge in den Ausgabe-String geschrieben werden.
Falls das aktuell gelesene Zeichen ein Operator-Zeichen ist, wird die folgende Regel angewandt: Falls bereits ein Operator-Zeichen auf dem Stack liegt, wird anhand der Operator-Rangfolge (Priorität, Präzedenz) und der Assoziativität des Operators entschieden, ob das neue Operator-Zeichen direkt auf den Stack gelegt wird, oder der Stack zuerst in den Ausgabe-String geleert und das neue Operator-Zeichen dann auf den Stack gelegt wird. Wenn die Präzedenz des neuen Operators niedriger ist als die des aktuellen Operators auf dem Stack und der Operator links-assoziativ ist, dann wird der Operator vom Stack genommen und in den Ausgabe-String geschrieben.
Öffnende Klammern werden ebenfalls auf den Stack gelegt, allerdings werden sie beim Entfernen nicht in den Ausgabe-String geschrieben. Wird eine schließende Klammer gelesen, dann wird der Stack bis zum Antreffen einer öffnenden Klammer in den Ausgabe-String geleert, und die öffnende Klammer wird gelöscht. Gehen Sie davon aus, dass es zu jeder schließenden Klammer eine geöffnete Klammer gibt, und umgekehrt. Schließende Klammern werden nicht in den Ausgabe-String geschrieben.
Verwollständigen Sie die gegebene Implementierung des Shunting-yard-Algorithmus (**infixToPostfix**), Hierzu verwenden Sie die Klasse Stack sowie die Objekt-Methoden **isEmpty()**: liefert true, wenn der Stack leer ist, und ansonsten false, **push()** und **pop()** wie in der Vorlesung besprochen, sowie **peek()**: liest das oberste Element des Stacks ohne es vom Stack zu löschen. Füllen Sie im folgenden Programm die Lücken an den gekennzeichneten Stellen. Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.
Ihren Code für die i-te Lücke schreiben Sie in der Form: i) Code

```
boolean letterOrDigit(char c) {

    if (Character.isLetterOrDigit(c))
        return true;
    else
        return false;
}

int getPrecedence(char ch) {

    if (ch == '+' || ch == '-')
        return 1;
    else if (ch == '*' || ch == '/')
        return 2;
    else
        return -1;
}

String infixToPostfix(String expression) {

    // Initialisiere einen leeren Stack und Ausgabe-String
    __________(1)______________

    __________(2)______________

    // Iteration über alle Zeichen der Eingabe
    for (int i = 0; i < __________(3)______________; ++i) {

        char c = expression.charAt(i);
```

```
        if (letterOrDigit(c)) {
            __________(4)_____________
        }
        else {
            if (c == '(')
                __________(5)_____________
            else {
                if (c == ')') {

                    while (!stack.isEmpty() && __________(6)_____________)
                        output += stack.pop();

                    __________(7)_____________
                }
                else {
                    // berücksichtige die Präzedenz
                    while (!stack.isEmpty() &&
                            __________(8)_____________ ) {

                        output += stack.pop();
                    }

                    __________(9)_____________
                }
            }
        }
    }

    // Alle Operatoren in die Ausgabe
    while (!stack.isEmpty()) {

        output += stack.pop();
    }

    return output;
}
```

Geben Sie hier ihre Lösungen an.

Aufgabe 4 Arrays (11 Punkte)

0 1 2 3 4 5 6 7 8 9

a) Vervollständigen Sie das folgende Programm, das ein Feld von Integer-Zahlen bzgl. der Zahlen in einem zweiten Integer-Feld sortiert. Die Zahlen im zweiten Feld geben die Reihenfolge an, in der die Zahlen im ersten Feld nach der Sortierung liegen. Gleiche Zahlen werden hintereinander angeordnet. Besitzt das erste Feld Zahlen, die nicht im zweiten Feld vorkommen, sollen diese Zahlen aufsteigend sortiert sein und nach den Zahlen liegen, die im zweiten Feld vorkommen.
Beispiel:
first = [5, 8, 9, 3, 5, 7, 1, 3, 4, 9, 3, 5, 1, 8, 4]
second = [3, 5, 7, 2]
Ausgabe: [3, 3, 3, 5, 5, 5, 7, 1, 1, 4, 4, 8, 8, 9, 9]
Verwenden Sie die Klasse TreeMap, deren aktuelle Einträge immer sortiert sind hinsichtlich der Ordnung ihrer Keys. Berücksichtigen sie, dass TreeMap das Interface Map implementiert, das insbesondere die folgenden Methoden bereitstellt:
default V getOrDefault(Object key, V defaultValue)
Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
V put(K key, V value)
Associates the specified value with the specified key in this map.
V remove(Object key) Removes the mapping for a key from this map if it is present.

Ihre Implementierung soll auf folgendem Prinzip basieren: Es wird eine TreeMap erstellt, die für jede Zahl des ersten Feldes dessen Auftretens-Häufigkeit speichert. Dann sollen die Zahlen in der Reihenfolge, die durch die Zahlen im zweiten Feld gegeben ist, in das erste Feld geschrieben werden. Die Häufigkeit gibt an, wie viele gleiche Zahlen jeweils nacheinander im Ausgabe-Feld stehen.
Nachdem die Aufgabe bearbeitet wurde, sind aus der Map alle Elemente gelöscht, die auch im zweiten Feld vorkommen. Sortieren sie nun die verbleibenden Elemente in aufsteigender Reihenfolge aus der Map in das erste Feld, ohne die bereits gespeicherten Elemente zu überschreiben.

```
public static void customSort(int[] first, int[] second) {

    Map<___(1)___, ___(2)___> freq = new TreeMap<>();

    // finde die Häufigkeit des Vorkommens von jedem
    // Element des ersten Feldes. Tipp: Verwenden sie
    // getOrDefault und put in Kombination
    for (int i: first) {
        __________(3)__________;
    }

    int index = 0;

    for (int i: second) {

        int n = freq.getOrDefault(i, 0);
        while (n-- > 0) {
            first[___(4)___] = i;
        }

        __________(5)__________;
    }

    // Sortiere die verbleibenden Elemente in das erste Feld

    for (___(6)___ : ___(7)___) {
        ________(8)________;
        while (count-- > 0) {
            ________(9)________;
        }
    }
}
```

Geben Sie hier Ihre Antworten an.

b)
Geben Sie die Ausgabe des folgenden Programms an. Sie können davon ausgehen, dass das Programm fehlerfrei übersetzt und ausgeführt wird.

0
1
2

```
public class Test {

    void merge(int[] arr, int[] aux, int low, int mid, int high) {

        int k = low;

        for (int i = low; i <= mid; i++) {
            if (arr[i] < 0) {
                aux[k++] = arr[i];
            }
        }

        for (int j = mid + 1; j <= high; j++) {
            if (arr[j] < 0) {
                aux[k++] = arr[j];
            }
        }

        for (int i = low; i <= mid; i++) {
            if (arr[i] >= 0) {
                aux[k++] = arr[i];
            }
        }

        for (int j = mid + 1; j <= high; j++) {
            if (arr[j] >= 0) {
                aux[k++] = arr[j];
            }
        }

        for (int i = low; i <= high; i++) {
            arr[i] = aux[i];
        }
    }


    void partition(int[] arr, int[] aux, int low, int high) {

        if (high <= low) {
            return;
```

```
        }

        int mid = (low + ((high - low) >> 1));

        partition(arr, aux, low, mid);
        partition(arr, aux, mid + 1, high);

        merge(arr, aux, low, mid, high);
    }


    public static void main(String[] s) {

        int[] myarr = { 9, -3, 5, -2, -8, -6, 1, 3 };
        int[] myaux = Arrays.copyOf(myarr, myarr.length);

        partition(myarr, myaux, 0, myarr.length - 1);

        System.out.println(Arrays.toString(myarr));
    }
}
```

Geben Sie hier Ihre Antwort an.

Aufgabe 5 Listen (13 Punkte)

a)
Gegeben sei die folgende Implementierung zum Mergen von k **sortierten** Listen, die in einem Feld von Listen gespeichert sind. Die zurückgelieferte Liste soll wiederum sortiert sein. Verwenden Sie die Klasse **Class PriorityQueue<E>**: The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time.
Sie können dann die folgenden Routinen nutzen:
public boolean add(E e) : Inserts the specified element into this priority queue.
public E poll() : Retrieves and removes the head of this queue, or returns null if this queue is empty.
Da die Klasse das **Interface Collection<E>** implementiert, stehen folgende Routinen zur Verfügung:
boolean isEmpty() : Returns true if this collection contains no elements.
boolean addAll(Collection<? extends E> c) : Adds all of the elements in the specified collection to this collection.

0 1 2 3 4 5 6 7 8

Ihre Implementierung soll auf dem folgenden Prinzip basieren:
Einfügen der ersten Elemente aus allen Listen in die PriorityQueue
Lesen und Löschen des aktuellen Minimums
Einfügen des nächsten Elements aus der Liste, aus der das Minimum stammt

Ergänzen Sie den Code der Methode **Node mergeKLists(Node[] lists)**, um k sortierte Listen zu mergen. Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.
Den Code für die i-te Lücke schreiben Sie in der Form: i) Code

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

Node mergeKLists(Node[] lists) {

    // Anlegen der priority-queue mit einem Vergleichs-Objekt zur Sortierung der Queue-Elemente.
    // comparingInt benötigt als Argument ein Objekt, welches das funktionale Interface
    // ToIntFunction<T> mit der zu implementierenden Funktion int applyAsInt(T value) erfüllt
    PriorityQueue<Node> pq;
    pq = new PriorityQueue<>(Comparator.comparingInt(______(1)______);

    // speichern aller ersten Element der Listen in die Queue
    pq.addAll(_______(2)_______);

    // head und tail sollen auf das erste bzw. letzte Element der Ausgabe-Liste zeigen
    Node head = null, last = null;

    while (_____(3)_____) {

        // hole das minimale Element aus der Queue
        Node min = ____(4)____;

        if (head == null) {
            ____(5)____;
        }
        else {
            ____(6)____;
        }

        // Füge das nächste Element aus der Liste ein, aus der das Minimum stammt
```

```
            if (____(7)____) {
                pq.add(____(8)____);
            }
        }

        // gebe die Output-Liste zurrück
        return head;
    }
```

Geben Sie hier Ihre Antworten an.

b)
Vervollständigen sie die folgenden Implementierungen der Routinen **leafTraversal(Node x)** und **Node getNextLeafNode(Stack<Node> s)**, so dass bei Aufruf von **leafTraversal(Node x)** mit dem Start-Knoten (vom Typ Node) eines Binärbaumes die Variable **leaf** nacheinander alle **Leaf-Nodes** des Binärbaumes in preorder-Traversierung beinhaltet. Preorder heißt, dass die Wurzel eines Baumes zuerst besucht wird, und dann der linke vor dem rechten Teilbaum traversiert wird. Für jeden Teilbaum wird diese Traversierung rekursiv wiederholt.

0 1 2 3 4 5

Beachten Sie, dass die Traversierung mit einem Stack implementiert werden muss. Hierzu werden die Knoten so auf den Stack gelegt, dass jeweils die Knoten des linken Teilbaums höher und die Knoten des rechten Teilbaums tiefer im Stack liegen. Wenn alle Knoten eines linken Teilbaums vom Stack geholt wurden, beginnt die Traversierung mit dem Start-Knoten des rechten Teilbaums. Benutzen Sie die aus der Vorlesung bekannten Stack-Methoden push, pop, empty. Beginnen Sie mit der Bearbeitung an der Methode **leafTraversal(Node x)**.

Im Code ist die i-te Lücke mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.
Den Code für die i-te Lücke schreiben Sie in der Form: i) Code

```
class Node {
    int data;
    Node left, right;

    public Node(int data) {
        this.data = data;
        this.left = this.right = null;
    }
}

// test, ob ein Knoten ein Leaf-Node ist oder nicht
boolean isLeaf(Node node) {
    return node.left == null && node.right == null;
}

// liefert den nächsten Leaf-Node eines Binärbaumes in preorder-Sortierung
Node getNextLeafNode(Stack<Node> s) {

    Node curr = s.pop();

    while (curr != null && ____(2)____) {

        if (curr.right != null) {
            ____(3)____;
        }

        if (curr.left != null) {
            ____(4)____;
        }

        ____(5)____;
    }

    // aktueller Knoten ist Leaf-Node
    return curr;
}

void leafTraversal(Node x) {

    Stack<Node> s = new Stack<>();

    // push the root node of the tree into the stack
    s.push(x);
```

```
    // loop bis stack leer
    while (____(1)_____) {

        // hole den nächsten leaf-Node in preorder-Sortierung aus tree
        Node leaf = getNextLeafNode(s);
    }
}
```

Geben Sie hier Ihre Antworten an.

Aufgabe 6 Streams, Collections und Interfaces (15 Punkte)

a) Das folgende Programm soll alle String-Elemente des gegebenen Feldes, die nur aus einem Zeichen bestehen, nacheinander in der Form ”Token: ” Zeichen ”;” ausgeben. Füllen Sie die Lücken entsprechend der Aufgabe mit Code. Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.
Den Code für die i-te Lücke schreiben Sie in der Form: i) Code

0 1 2 3 4

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

        List<String> list = Arrays.asList("a", "bb", "ccc", "dd", "x", "zz", "u");

        list.stream()
            ___________(1)___________
            ___________(2)___________
            .forEach(i -> System.out.print(i));
    }
}
```

Geben Sie hier ihre Lösungen an.

b)
Nehmen Sie an, dass das funktionale Interface GenericInterface wie folgt definiert ist:

0 1 2 3

```
@FunctionalInterface
interface GenericInterface<T> {

    // generic method
    T func(T t);
}
```

Füllen Sie die Lücken im folgenden Java-Code, so dass folgende Ausgabe vom Programm generiert wird:
Lambda O-O = OLambdaO

Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.

```
public class HelloWorld {
    public static void main(String[] s) {

    ___________(1)___________ =

        ________(2)________ {

            __________(3)__________
        };

        System.out.println("Lambda O-O = " + modify.func("Lambda"));
    }
}
```

Geben Sie hier ihre Lösungen an.

cit-eidi-1-20240212-E0576-21

0 1 2 3 4

c) Geben Sie die Ausgaben der beiden folgenden Java-Statements an, unter der Annahme, dass die Streams nicht parallel verarbeitet werden.

```
(1) Stream.of("d2", "a2", "b1")
    .filter(s -> {
        System.out.println("filter: " + s);
        return true;
    });

(2) Stream.of("d2", "a2", "b1", "b3", "c")
    .filter(s -> {
        System.out.println("filter: " + s);
        return s.startsWith("a");
    })
    .map(s -> {
        System.out.println("map: " + s);
        return s.toUpperCase();
    })
    .forEach(s -> System.out.println("forEach: " + s));
```

Geben Sie hier ihre Lösungen an.

0 1 2 3 4

d) Gegeben ist das folgende unvollständige Java-Programm. Gehen Sie davon aus, dass die nicht implementierten Methoden die richtigen Ergebnisse liefern.

```
class Person {

    public enum Sex {
        MALE, FEMALE
    }

    String name;
    LocalDate birthday;
    Sex gender;
    String emailAddress;

    public int getAge() {
        //...
    }

    public void printPerson() {
```

```
        // ...
    }

    public String getEmailAddress() {
        // ...
    }
}

public class HelloWorld {
    public static void main(String[] s) {

    List<Person> personList;

    //... hier wird personList mit Personen gefüllt
    addPersons(personList);

    processElements (
        personList,
        p -> p.getGender() == Person.Sex.MALE
        && p.getAge() >= 18
        && p.getAge() <= 25,
        p -> p.getEmailAddress(),
        email -> System.out.println(email)
    );
}
```

Ergänzen Sie die folgende Implementierung der Methode **processElements**, so dass der Aufruf der Methode im Programm die Email-Adressen aller männlichen Personen im Alter von 18 bis 25 Jahren ausgibt. Füllen Sie die Lücken im folgenden Java-Code.
Die i-te Lücke ist mit (i) nummeriert. Schreiben Sie Ihre Lösungen für die Lücken untereinander in der Reihenfolge der Nummerierung in das Textfeld.

```
public static <X, Y> void processElements (
    Iterable<X> source,
    Predicate<X> tester,
    Function <X, Y> mapper,
    Consumer<Y> block) {
    _______(1)_______
    _______(2)_______
    _______(3)_______;
    _______(4)_______;
    }
    }
}
```

Geben Sie hier ihre Lösungen an.

Zusätzlicher Platz für Lösungen. Markieren Sie deutlich die Zuordnung zur jeweiligen Teilaufgabe. Vergessen Sie nicht, ungültige Lösungen zu streichen.