

Aufgabe 1:

Suchen Sie sich vier Aufgaben aus der bisherigen Vorlesung aus. Stellen Sie diese Aufgaben als Anfragen an ChatGPT. Dokumentieren Sie diese und prüfen Sie die Plausibilität/Korrektheit der Lösung.

Sollten Sie im ersten Anlauf nicht mit der Antwort einverstanden sein, so formulieren Sie Ihren Prompt bitte um. Befragen Sie ChatGPT ein zweites und ggf. ein drittes Mal!

1. Aufgabe 04-03 (Vorlesung-Übungsaufgabe)

Sieb des Erathostenes

1.1. Prompt

Bitte löse folgende Aufgabe in Java

- Aufgabe 3

Das Sieb des Erathostenes (nach Eratosthenes von Cyrene ca. 276 - 194 v.u.Z.) ist ein Algorithmus zur Bestimmung aller Primzahlen zwischen 1 und einer festgelegten oberen Schranke. Erathostenes ging dabei (damals noch ohne Computer) wie folgt vor:

1. Er schrieb alle Zahlen von 1 bis s in einer Reihe auf.

Beispielsweise mit $s = 15$: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

2. Er nahm zuerst die 2 als Basis und strich alle ihre Vielfachen durch:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

3. Dann nahm er die nächste noch nicht durchgestrichene Zahl (also zunächst die 3) als

Basis und strich alle ihre Vielfachen durch:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

4. Er überprüfte, ob die letzte Zahl, die er als Basis gewählt hatte, größer als $s/2$ war. Falls nein, wiederholte er Schritt 3 mit der nächsten noch nicht durchgestrichenen Zahl.

Falls ja, war er fertig.

Die noch nicht durchgestrichenen Zahlen entsprechen genau der Menge der Primzahlen zwischen 1 und s .

Entwickeln Sie ein Programm, das das Sieb des Erathostenes anwendet, um alle Primzahlen zwischen 1 und einer Integer-Konstante `MAXIMUM` auszugeben!

1.2. Antwort

```
public class SiebDesErathostenes {  
  
    public static void main(String[] args) {  
  
        final int MAXIMUM = 100; // Beispielwert, kann angepasst werden  
  
        boolean[] isPrime = new boolean[MAXIMUM + 1];  
  
    }  
}
```

```
    for (int i = 2; i <= MAXIMUM; i++) {  
        isPrime[i] = true;  
    }  
    for (int i = 2; i * i <= MAXIMUM; i++) {  
        if (isPrime[i]) {  
            for (int j = i * i; j <= MAXIMUM; j += i) {  
                isPrime[j] = false;  
            }  
        }  
    }  
    System.out.println("Primzahlen zwischen 1 und " + MAXIMUM + ":");  
    for (int i = 2; i <= MAXIMUM; i++) {  
        if (isPrime[i]) {  
            System.out.print(i + " ");  
        }  
    }  
}
```

Dieses Programm implementiert das Sieb des Eratosthenes, um alle Primzahlen zwischen 1 und einer festgelegten oberen Schranke (in diesem Fall die Integer-Konstante MAXIMUM) auszugeben. Das Array isPrime wird verwendet, um zu speichern, ob eine Zahl als prim markiert ist oder nicht. Zuerst werden alle Zahlen von 2 bis MAXIMUM als prim markiert. Dann wird das Sieb des Eratosthenes angewendet, um alle Vielfachen der primes zu markieren. Schließlich werden alle prim markierten Zahlen ausgegeben.

1.3. Bewertung

Das Programm ist ohne Korrekturen lauffähig und liefert korrekte Ergebnisse. Obergrenzen innerhalb der for-Schleifen können zur Vermeidung unnötiger Rechenschritte angepasst werden, um das Programm weiter zu optimieren.

2. Aufgabe 04-05
Der Goldfrosch

2.1. Prompt

Löse bitte alle Teilaufgaben dieser Übung:

Aufgabe 5

Der Goldfrosch ist ein virtuelles Tier, das sich auf der Zahlengerade mit einer Schrittw

eite von +1 oder -2 bewegen kann. An jeder Stelle $i = 1, 2, 3, \dots$ ist ein Richtungspfeil angebracht, der nach rechts oder links weist. Landet ein Frosch auf Platz i , so dreht er den

Richtungspfeil zunächst um und hüpft dann in der neuen Richtung, wobei folgende Regeln gelten: Zeigt der Pfeil nach links, hüpft der Frosch von Platz i nach $i-2$, andernfalls

von Platz i nach Platz $i+1$. Auf Platz 0 und Platz -1 steht jeweils ein Eimer, in den der Frosch plumpst, wenn er zu weit nach links abgekommen ist. In diesem Fall startet ein neuer Frosch auf Platz 1. Zu Beginn weisen alle Richtungspfeile nach rechts.

a) Schreiben Sie ein Java-Programm (unter Verwendung der bisher bekannten Sprachmittel), das den Froschprozess nachvollzieht und am Ende die Anzahl der Frösche in den beiden Eimern angibt!

b) Wie ist das Verhältnis der Frosch-Anzahlen in beiden Eimern auf lange Sicht?

c) Kann es vorkommen, dass ein Frosch nach rechts ins Unendliche wegspringt?

2.2. Antwort

Hier ist eine mögliche Lösung in Java für die Teilaufgabe a):

```
public class Goldfrosch {  
  
    public static void main(String[] args) {  
  
        int numFrogsInLeftEimer = 0;  
  
        int numFrogsInRightEimer = 0;  
  
        int currentPosition = 1;  
  
        int direction = 1; // 1: nach rechts, -1: nach links  
  
        for (int i = 1; i <= 1000; i++) { // Wir lassen die Frösche für 1000  
            Schritte hüpfen  
  
            if (currentPosition == 0 || currentPosition == -1) { // Der Frosch  
                ist im Eimer gelandet  
  
                if (currentPosition == 0) {
```

```
        numFrogsInLeftEimer++;
    } else {
        numFrogsInRightEimer++;
    }

    currentPosition = 1; // Ein neuer Frosch startet auf Platz 1
    direction = 1; // Pfeil zeigt zu Beginn immer nach rechts
}

if (direction == 1) {
    currentPosition += 1;
    direction = -1; // Pfeil ändern
} else {
    currentPosition -= 2;
    direction = 1; // Pfeil ändern
}

}

    System.out.println("Anzahl der Frösche im linken Eimer: " +
numFrogsInLeftEimer);

    System.out.println("Anzahl der Frösche im rechten Eimer: " +
numFrogsInRightEimer);

}

}
```

Dieses Programm simuliert den Froschprozess für 1000 Schritte und zählt dann die Anzahl der Frösche in den beiden Eimern.

b) Das Verhältnis der Froschanzahlen in beiden Eimern auf lange Sicht ist 1:1. Dies liegt daran, dass der Frosch immer um eins nach rechts oder um zwei nach links hüpfte. Da der Froschprozess solange wiederholt wird, bis ein neuer Frosch auf den Platz 1 gesetzt wird, wird sich die Anzahl der Frösche in beiden Eimern angleichen.

c) Nein, es kann nicht vorkommen, dass ein Frosch nach rechts ins Unendliche weghüpft. Da die Schritte des Frosches begrenzt sind (entweder +1 oder -2), wird der Froschprozess in einem begrenzten Bereich bleiben.

2.3. Korrektur 1

2.3.1.Prompt

Die Pfeile zeigen zu Beginn der gesamten Simulation nach rechts. Die Konfiguration, die der vorangegangene Frosch hinterlässt, wird weiterverwendet. Bitte beachte dies bei der Korrektur aller 3 Teilaufgaben

2.3.2.Antwort

Hier ist die überarbeitete Lösung unter Berücksichtigung der Tatsache, dass die Pfeile zu Beginn der gesamten Simulation nach rechts zeigen und die Konfiguration, die der vorangegangene Frosch hinterlässt, weiterverwendet wird:

```
public class Goldfrosch {  
  
    public static void main(String[] args) {  
  
        int numFrogsInLeftEimer = 0;  
  
        int numFrogsInRightEimer = 0;  
  
        int currentPosition = 1;  
  
        for (int i = 1; i <= 1000; i++) { // Wir lassen die Frösche für 1000  
Schritte hüpfen  
  
            if (currentPosition == 0 || currentPosition == -1) { // Der Frosch  
ist im Eimer gelandet  
  
                if (currentPosition == 0) {  
  
                    numFrogsInLeftEimer++;  
  
                } else {  
  
                    numFrogsInRightEimer++;  
  
                }  
  
                currentPosition = 1; // Ein neuer Frosch startet auf Platz 1  
  
            } else if (currentPosition == 1) {  
  
                currentPosition = -2; // Pfeil zeigt nach links, hüpfte nach i-2
```

```
        } else {  
            currentPosition = currentPosition + 1; // Pfeil zeigt nach  
rechts, hüpfte nach i+1  
        }  
    }  
  
    System.out.println("Anzahl der Frösche im linken Eimer: " +  
numFrogsInLeftEimer);  
  
    System.out.println("Anzahl der Frösche im rechten Eimer: " +  
numFrogsInRightEimer);  
}  
}
```

Mit dieser Anpassung wird die Erklärung sowie das Programm in Anlehnung an die Aufgabenstellung korrigiert.

Die Aussagen zu b) und c) ändern sich nicht, die Erklärungen bleiben korrekt. Das Verhältnis der Froschanzahlen in beiden Eimern bleibt 1:1, und es kann nicht vorkommen, dass ein Frosch nach rechts ins Unendliche wegehüpft.

2.4. Korrektur 2

2.4.1. Prompt

Die Ergebnisse sind falsch. geliefert wird 0 im Linken und 333 im rechten Eimer. Bei 1000 kann das so nicht stimmen.

2.4.2. Antwort

Entschuldigung für den Fehler. Hier ist die überarbeitete Lösung unter Berücksichtigung der Anmerkungen:

```
public class Goldfrosch {  
  
    public static void main(String[] args) {  
  
        int numFrogsInLeftEimer = 0;  
  
        int numFrogsInRightEimer = 0;  
  
        int currentPosition = 1;
```

```
        for (int i = 1; i <= 1000; i++) { // Wir lassen die Frösche für
1000 Schritte hüpfen

            if (currentPosition == 0 || currentPosition == -1) { //
Frosch landet im Eimer

                if (currentPosition == 0) {

                    numFrogsInLeftEimer++;

                } else {

                    numFrogsInRightEimer++;

                }

                currentPosition = 1; // Ein neuer Frosch startet auf
Platz 1

            }

            currentPosition = getPositionAfterHop(currentPosition);

        }

        System.out.println("Anzahl der Frösche im linken Eimer: " +
numFrogsInLeftEimer);

        System.out.println("Anzahl der Frösche im rechten Eimer: " +
numFrogsInRightEimer);

    }

    public static int getPositionAfterHop(int currentPos) {

        if(currentPos == 1) {

            return -2;

        } else {

            return currentPos+1;

        }

    }

}
```

Mit dieser Anpassung sollte die Anzahl der Frösche in den Eimern korrekt berechnet werden, und zwar 0 im linken Eimer und 667 im rechten Eimer.

2.5. Bewertung

Die Programme sind alle lauffähig, jedoch nicht korrekt. Die Ergebnisse sind auch nach den Korrekturen deutlich falsch. So wird in allen Versionen für einen Eimer immer 0 ausgegeben.

3. Matrizen

3.1. Prompt

Löse die folgende Aufgabe

Aufgabe 2

Es soll Software zum Arbeiten mit reellen Matrizen erstellt werden.

Folgendes soll möglich sein:

- Anlegen von Matrizen über einen Konstruktor, dem als Parameter die Anzahl der Zeilen und Spalten übergeben werden
- Ausgeben der Matrizen
- Setzen und Ausgeben von Attributen soll über set- und get-Methoden erfolgen
- Skalarmultiplikation, d.h. Multiplikation der Matrix mit einer Zahl
- Addition zweier Matrizen
- Legen Sie in der main-Methode Objekte an und testen Sie anhand derer ihre Implementierung

3.2. Antwort

Hier ist eine mögliche Implementierung in Java:

```
public class Matrix {  
  
    private int[][] matrix;  
  
    private int rows;  
  
    private int columns;  
  
    public Matrix(int rows, int columns) {  
  
        this.rows = rows;
```



```
        this.columns = columns;

        this.matrix = new int[rows][columns];
    }

    public void setElement(int row, int column, int value) {
        this.matrix[row][column] = value;
    }

    public int getElement(int row, int column) {
        return this.matrix[row][column];
    }

    public void scalarMultiplication(int scalar) {
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.columns; j++) {
                this.matrix[i][j] *= scalar;
            }
        }
    }

    public Matrix add(Matrix otherMatrix) {
        if (this.rows != otherMatrix.rows || this.columns !=
otherMatrix.columns) {
            throw new IllegalArgumentException("Matrices must have the same
dimensions");
        }

        Matrix result = new Matrix(this.rows, this.columns);

        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.columns; j++) {
```

```
        result.setElement(i, j, this.matrix[i][j] +
otherMatrix.getElement(i, j));
    }
}
return result;
}
```

```
public void printMatrix() {
    for (int i = 0; i < this.rows; i++) {
        for (int j = 0; j < this.columns; j++) {
            System.out.print(this.matrix[i][j] + " ");
        }
        System.out.println();
    }
}
```

```
public static void main(String[] args) {
    Matrix m1 = new Matrix(2, 2);
    m1.setElement(0, 0, 1);
    m1.setElement(0, 1, 2);
    m1.setElement(1, 0, 3);
    m1.setElement(1, 1, 4);
    System.out.println("Matrix m1:");
    m1.printMatrix();

    m1.scalarMultiplication(2);
    System.out.println("Matrix m1 after scalar multiplication:");
    m1.printMatrix();
}
```

```
Matrix m2 = new Matrix(2, 2);

m2.setElement(0, 0, 5);

m2.setElement(0, 1, 6);

m2.setElement(1, 0, 7);

m2.setElement(1, 1, 8);

System.out.println("Matrix m2:");

m2.printMatrix();

Matrix m3 = m1.add(m2);

System.out.println("Matrix m3 = m1 + m2:");

m3.printMatrix();

}

}
```

3.3. Bewertung

Das Programm erfüllt die Anforderungen und ist bereits beim ersten Versuch korrekt. Es wurde beachtet dass nur Matrizen kompatibler Formate addiert werden können und dementsprechend eine Exception verbaut um die Ausnahmefälle zu behandeln.

Aufgabe 2:

1. Strings runden

Ziel ist es ein Programm zu schreiben das in der Lage ist anhand einer eigenen Implementation Zahlen auf eine bestimmte Anzahl an Nachkommastellen zu runden.

Input ist ein String der eine Gleitkomma- oder auch ganze Zahle enthalten kann und die gewünschte Zahl an Nachkommastellen als Integer Wert.

Der Output soll ebenfalls vom Datentyp String sein.

1.1. Input

Löse folgende Aufgabe
[siehe 1.]

1.2. Output

Ja, natürlich. Hier ist eine Implementierung ohne die Verwendung von anderen Paketen:

```
public class Rounding {

    public static String roundToNDecimalPlaces(String number, int
decimalPlaces) {

        int index = number.indexOf('.');

        if (decimalPlaces == 0) {

            return number.substring(0, index); // return the whole number
before the decimal point

        } else {

            if (index == -1) { // if the number is an integer

                return number;

            } else {

                String roundedNumber = number.substring(0, index +
decimalPlaces + 1); // get the number including the specified decimal places

                return roundedNumber;

            }

        }

    }

    public static void main(String[] args) {

        String input = "3.14159";

        int decimalPlaces = 2;

        String result = roundToNDecimalPlaces(input, decimalPlaces);

        System.out.println("Rounded number: " + result);

    }

}
```

In dieser Implementierung wird die Methode `roundToNDecimalPlaces` verwendet, um die Eingabe auf die gewünschte Anzahl von Nachkommastellen zu runden. Der gerundete Wert wird als String zurückgegeben und kann dann weiterverwendet oder ausgegeben werden.

1.3. Bewertung

Im ersten Moment scheinen die Ergebnisse korrekt zu sein, jedoch fällt beispielsweise bei der Input-Kombination („3.14159“, 3) auf, dass der Output von „3.141“ nicht mathematisch korrekt aufgerundet wurde.

2. Primfaktorzerlegung

Gegeben ist eine ganze Zahl N .

Der Gewünschte Output ist eine Liste der Primfaktoren von N .

D.H. das Produkt aller in der Liste enthaltenen Zahlen muss exakt N sein.

Dabei dürfen nur Primzahlen in der Liste enthalten sein.

2.1. Prompt

Bitte erstelle eine Lösung dieser Aufgabe in Java:
[siehe 2.]

2.2. Output

Hier ist eine mögliche Implementierung in Java:

```
import java.util.ArrayList;
import java.util.List;

public class PrimeFactorization {
    public static List<Integer> primeFactorization(int n) {
        List<Integer> factors = new ArrayList<>();

        for (int i = 2; i <= n; i++) {
            while (n % i == 0) {
                factors.add(i);
                n /= i;
            }
        }
    }
}
```

```
        if (n > 2) {  
            factors.add(n);  
        }  
  
        return factors;  
    }  
  
    public static void main(String[] args) {  
        int number = 84;  
        List<Integer> primeFactors = primeFactorization(number);  
        System.out.println("Prime factors of " + number + ": " + primeFactors);  
    }  
}
```

Die Methode `primeFactorization` verwendet eine Schleife, um die Primfaktoren der gegebenen Zahl zu finden und sie in der Liste `factors` zu speichern. Schließlich gibt die `main`-Methode die Liste der Primfaktoren aus.

2.3. Bewertung

Die Implementierung scheint korrekt zu sein, mehrere Stichproben zeigten die richtigen Ergebnisse.

Aufgabe 3:

Prompt:

Gib mir eine Liste mit Ideen für verschiedene Programmier-übungen. Enthalten sein sollten Themen wie grundlegende Kontrollstrukturen, Rekursion und Objektorientierung.

1. Kryptographie

Implementierung der Caesar-Verschlüsselung oder anderer einfacher Chiffren.

Ver- und Entschlüsselungsalgorithmen sind interessant und in sehr vielen Varianten und Komplexitäten verfügbar. Dabei ist die Caesar Chiffre eine sehr simple Version die sich eher für Anfänger eignet.

Ich würde die Schwierigkeit als ähnlich zum „Sieb des Erathostenes“ aus Vorlesung 4 einstufen.

2. Passwortgenerator / Manager

Baue einen Passwortmanager, der Benutzern ermöglicht, sichere Passwörter zu generieren, zu speichern und zu verwalten. Dies erfordert Kenntnisse in der sicheren Speicherung von Passwörtern, Verschlüsselung, Datenbankmanagement und Benutzerschnittstellen.

Auch hier ist eine Vielzahl von Ausbaustufen möglich. Man könnte soweit gehen und die Idee als vollständige Applikation entwickeln oder als kleineres Programm das die Passwörter nur in .txt Dateien speichert. Abhängig vom Niveau des Programmierers kann so die richtige Schwierigkeit gewählt werden.

3. Merge-sort

Rekursive Sortieralgorithmen: Implementiere rekursive Sortieralgorithmen wie den Quicksort oder Mergesort. Diese Übung könnte ein tiefes Verständnis von rekursivem Algorithmusdesign und effizienten Sortiertechniken vermitteln.

Merge-sort ist ein klassischer Sortieralgorithmus der das Prinzip der Rekursion verwendet. Seine Funktionsweise ist gut verständlich, dadurch kann der Fokus völlig auf die Implementierung verschoben werden. Gerade Anfangs kann Rekursion sehr anspruchsvoll sein. Daher halte ich auch diesen Vorschlag für eine gute Idee.