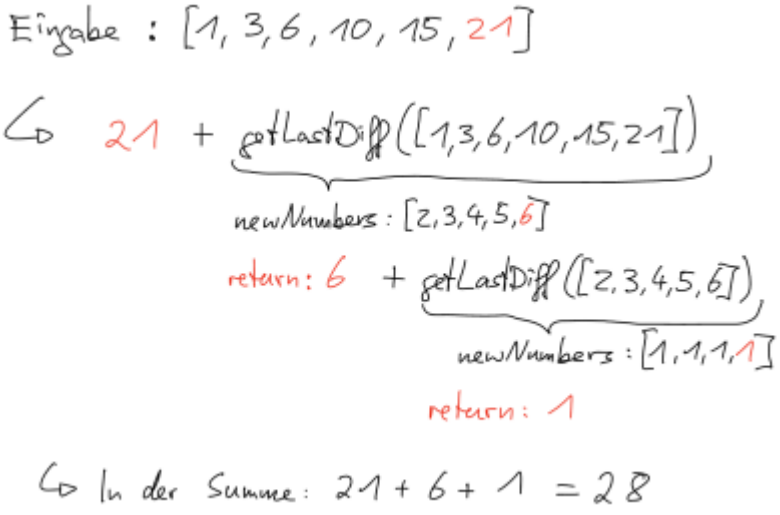
Tag 9: Mirage Maintenance

- Variante 1

Hilfestellung Teil 1

- Während es beim gestrigen Tag bei Verwendung von **Rekursion** zu einem Stack-Overflow kam, ist das heute kein Problem, da die Anzahl an rekursiven Aufrufen direkt linear von der übersichtlichen Anzahl der Zahlen pro Zeile abhängt.
- Es genügt eine einzige Schleife, in der du zunächst die aktuelle Zeile der Input-Datei in ein **long**-Array (z. B. `numbers`) umwandelst.
- Anschließend erhöhst du die Gesamtsumme um 'letzte Zahl von `numbers`' + `getLastDiff(numbers)`
- `getLastDiff(long[] lastNumbers)` wiederum bekommt nun als Parameter die jeweils letzte Zahlenfolge.
  - Als Erstes muss also in der Methode ein neues long-Array `newNumbers` angelegt werden, welches alle Differenzen zwischen den Werten von `lastNumbers` enthält.
  - Anschließend muss überprüft werden, ob `newNumbers` nur noch identische Zahlen enthält. Dann können wir nämlich bereits eine Zeile vor der Nuller-Zeile aufhören. Denn es ist klar, dass die nächsten Differenzen von identischen Zahlen nur noch 0 sein werden. Wenn dies also der Fall ist, dann ist die Abbruchbedingung erreicht und wir geben die Differenz zurück.
  - Andernfalls geben wir die 'letzte Zahl von `newNumbers`' + `getLastDiff(newNumbers)` zurück.

Der Rekursionsablauf sieht also in etwa so aus (zweites Bsp. der Aufgabe):



Lösungsvorschlag

```
public long partOne() {
    long summe = 0;

    for (String line: inputLines) {
        String[] numbersAsStr = line.split(" ");
```

```
        long[] numbers = new long[numbersAsStr.length];
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = Long.parseLong(numbersAsStr[i].trim());
        }

        summe += numbers[numbers.length - 1] + getLastDiff(numbers);
    }

    return summe;
}

private long getLastDiff(long[] lastNumbers) {
    long[] newNumbers = new long[lastNumbers.length - 1];

    boolean allNumbersTheSame = true;
    for (int i = 0; i < lastNumbers.length - 1; i++) {
        newNumbers[i] = lastNumbers[i+1] - lastNumbers[i];
        if (i > 0 && newNumbers[i] != newNumbers[i-1]) {
            allNumbersTheSame = false;
        }
    }

    if (allNumbersTheSame) {
        return newNumbers[0];
    }

    return newNumbers[newNumbers.length - 1] + getLastDiff(newNumbers);
}
```

Hilfestellung Teil 2

Für den Teil 2 müssen nun minimale Änderungen vorgenommen werden. Die Rechnung lautet nun nicht mehr 'letzte Zahl von `newNumbers`' + `getLastDiff(newNumbers)` sondern 'erste Zahl von `newNumbers`' - `getLastDiff(newNumbers)`. Dies muss sowohl beim ersten Rekursionsaufruf, als auch in der Rekursion geändert werden.

Lösungsvorschlag

```
public long partTwo() {
    long summe = 0;

    for (String line: inputLines) {
        String[] numbersAsStr = line.split(" ");
        long[] numbers = new long[numbersAsStr.length];
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = Long.parseLong(numbersAsStr[i].trim());
        }
```

```
        summe += numbers[0] - getFirstDiff(numbers);
    }

    return summe;
}

private long getFirstDiff(long[] lastNumbers) {
    long[] newNumbers = new long[lastNumbers.length - 1];

    boolean allNumbersTheSame = true;
    for (int i = 0; i < lastNumbers.length - 1; i++) {
        newNumbers[i] = lastNumbers[i+1] - lastNumbers[i];
        if (i > 0 && newNumbers[i] != newNumbers[i-1]) {
            allNumbersTheSame = false;
        }
    }

    if (allNumbersTheSame) {
        return newNumbers[0];
    }

    return newNumbers[0] - getFirstDiff(newNumbers);
}
```

From:
https://www.info-bw.de/ -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day9:start

Last update: **10.12.2023 17:05**