Day 7: Bridge Repair

Der heutige Tag lässt sich mit **Rekursion** relativ einfach lösen.

Teil 1

Tipps zur Vorgehensweise

- Du kannst direkt mit jeder einzelnen Zeile arbeiten und musst nicht zu Beginn zunächst alle Daten einlesen und in einer geeigneten Datenstruktur zwischenspeichern. Mache pro Zeile das Folgende:
- Die Zahlen werden/sind teilweise sehr groß → Arbeite mit `long` anstatt mit `int`, wo nötig!
- Speichere dir das erwartete Ergebnis. Tipp: `Long.parseLong(…)` um einen String in long zu konvertieren. Außerdem: `line.split(":")[0]`, um die Zahl **vor** dem Doppelpunkt zu bekommen.
- Speichere dir außerdem alle Zahlen rechts des Doppelpunkts in einer ArrayList von Integern. Tipp: Nutze auch hier die split-Methode, um zunächst alle Zahlen rechts des Doppelpunkts zu bekommen. Anschließend kann die Methode `trim()` sinnvoll sein, die ganz links und rechts vorhandene Leerzeichen löscht. Anschließend muss du die einzelnen Zahlen noch an den Leerzeichen aufsplitten. Diese Zahlen musst du nun noch von String zu Integer konvertieren und dann in der ArrayList speichern.
- Jetzt beginnt ein rekursiver Aufruf, mit dem du prüfst, ob auch nur eine "mathematische Kombination" mit +/* möglich ist, die zum gewünschten Ergebnis führt.
    - Gib als Parameter folgendes mit:
        - Das erwartete Ergebnis.
        - Die ArrayList aller Zahlen.
        - Ein Index, der angibt, die wievielte Zahl als nächstes zur Berechnung hinzugefügt werden muss.
        - Das aktuelle Ergebnis der Berechnung, das alle Zahlen von 0 bis zum Index "beinhaltet".
    - Was ist die Abbruchbedingung der Rekursion? Wann genau muss die Rekursion überprüfen, ob das "Ergebnis" korrekt ist und was muss daraufhin zurückgegeben werden?

Genau Vorgehensweise in der Rekursion:

- Zunächst die Abbruchbedingung: Wenn der Index gleich groß ist, wie die Anzahl der Zahlen in der ArrayList, dann bedeutet das, dass bereits alle Zahlen in die aktuelle Berechnung eingeflossen sind. Damit kann nun das Ergebnis überprüft werden. Das Ergebnis dieser Überprüfung (Vergleich der Berechnung mit dem erwarteten Ergebnis) ist auch direkt der Rückgabewert: Entweder es stimmt überein (true) oder nicht (false).
- Wenn der Index noch 0 ist, die Rekursion also gerade das erste Mal aufgerufen wird, dann müssen direkt zwei Zahlen miteinander verrechnet werden. Man prüft zunächst rekursiv, ob eine Addition der ersten beiden Zahlen korrekt ist. Der Index im nächsten rekursiven Aufruf muss also direkt auf 2 gesetzt werden, da gleich zwei Zahlen zu Beginn verrechnet werden müssen. Außerdem wird diese Rechnung `Zahl1 + Zahl2` als letzter Parameter gesetzt. Wenn

dieser erste Rekursive Aufruf mit der Addition nicht zum Erfolg geführt hat, dann muss das Ganze noch mit der Multiplikation durchgeführt werden.
- Wenn der Index >= 2 ist ("else"), da müssen ebenso nacheinander wieder die Fälle für die Addition und dann noch (falls nötig) für die Multiplikation überprüft werden. Nun muss aber jeweils der Index nur noch um 1 erhöht werden und das aktuelle Ergebnis der Berechnung ist "das aktuelle Ergebnis +/* der Zahl am gegebenen Index".

Lösungsvorschlag

```
public void partOne() {
    long sum = 0;

    for (String line: inputLines) {
        long result = Long.parseLong(line.split(":")[0]);

        ArrayList<Integer> numbers = new ArrayList();
        // Interiere über jede Zahl (noch als String-Array)
        for (String number: line.split(":")[1].trim().split(" ")) {
            numbers.add(Integer.parseInt(number));
        }

        if (equationPossible(result, numbers, 0, 0)) {
            sum += result;
        }
    }

    System.out.println(sum);
}

private boolean equationPossible(long result, ArrayList<Integer> numbers,
int index, long calculation) {
    // Abbruchbedingung: Alle Zahlen verrechnet
    if (index == numbers.size()) {
        return result == calculation;
    }

    if (index == 0) {
        if (equationPossible(result, numbers, 2, numbers.get(0) +
numbers.get(1))) {
            return true;
        } else if (equationPossible(result, numbers, 2, numbers.get(0) *
numbers.get(1))) {
            return true;
        }
    } else {
        if (equationPossible(result, numbers, index+1, calculation +
numbers.get(index))) {
            return true;
```

```
        } else if (equationPossible(result, numbers, index+1, calculation *
numbers.get(index))) {
            return true;
        }
    }
    return false;
}
```