Tag 14 - Parabolic Reflector Dish

- Variante 1

Der heutige Tag war wieder verhältnismäßig einfach, wenn man im Hinterkopf hat, wie man bereits Tag 12 lösen konnte.

Lösungshinweise Teil 1

- Speichere die Eingabe als zweidimensionales Character-Array.
- Rufe dann eine Methode auf, die dafür sorgt, dass jedes O soweit nach oben wie möglich bewegt wird:
  - Erzeuge ein neues, gleich großes char-Array, in welches du die aktualisierten Werte einträgst.
  - Gehe spaltenweise über das ursprüngliche Array, fange mit der linken Spalte an. Du gehst von **oben nach unten** über jede Zelle.
    - Wenn an der Position (x,y) # oder `.` gespeichert ist, dann wird dies genau so ins neue Array über- bzw. eingetragen.
    - Wenn an der Position (x,y) O gespeichert ist, dann musst du für alle darüberliegenden Werte im neuen Array prüfen, ob O auch dort gespeichert werden könnte. Lege dir dafür ein dx an, welches bei x beginnt und so lange reduziert wird (nach oben geht) wie dort ein `.` gespeichert ist. Speichere dann bei (dx,y) das O und stelle sicher, dass bei (x,y) ein `.` gespeichert ist. Dann geht es wieder regulär beim nächsten x weiter.
- Am Ende iterierst du wieder über alle Zellen. Wenn die Zelle O enthält, dann addierst du den Zeilenwert zur Summe.

Lösungsvorschlag

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

    // Lies alles in ein char[][] ein
    char[][] input = new
char[inputLines.get(0).length()][inputLines.size()];
    for (int y = 0; y < inputLines.size(); y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < line.length(); x++) {
            input[x][y] = line.charAt(x);
        }
    }

    // alles nach oben bewegen
    input = moveUp(input, inputLines.size(), inputLines.get(0).length());

    // gehe über jeden Character c und addiere seinen Wert zur Summe, wenn c
== 'O'
    for (int y = 0; y < inputLines.size(); y++) {
        for (int x = 0; x < inputLines.get(0).length(); x++) {
```

```
            if (input[x][y] == 'O') {
                summe += inputLines.size() - y;
            }
        }
    }

    return summe;
}

private char[][] moveUp(char[][] input, int hoehe, int breite) {
    char[][] output = new char[breite][hoehe];

    for (int x = 0; x < breite; x++) {
        for (int y = 0; y < hoehe; y++) {
            if (input[x][y] == '#') {
                output[x][y] = '#';
            } else if (input[x][y] == '.') {
                output[x][y] = '.';
            } else if (input[x][y] == 'O') {
                // bubble up (wie eine Blase nach oben bewegen ;-) )
                int dy = y;
                while (dy > 0 && output[x][dy-1] == '.') {
                    dy--;
                }
                output[x][dy] = 'O';
                // fülle das eigentliche y mit '.' auf, falls das 'O'
verschoben wurde
                if (dy != y) {
                    output[x][y] = '.';
                }
            }
        }
    }
    return output;
}
```

Lösungshinweise Teil 2

- Lege noch zwei weitere Methoden für die Richtungen links, unten und rechts an! Du kannst bei der bisherigen Methode copy-pasten, musst nur beim Anpassen der x-/y-Werte bzw. der dx-/dy-Werte aufpassen.
- Würdest du alle 1 Milliarde Rotationen ausrechnen, dann würde es Ewigkeiten dauern! Die Wahrscheinlichkeit ist ja aber sehr groß, dass sich nach ein paar Rotationen schließlich eine Situation eingefunden hat, die nach ein paar weiteren Rotationen exakt wieder so vorkommt (und so ist es auch!). Du landest also schließlich in einer Art Schleife, sodass immer wieder nach x-Rotationen die gleiche Ausgangslage gefunden ist. Das machen wir uns zunutze und speichern uns nach jeder Rotation den exakten Zustand und die Zahl des Durchlaufs / der Rotation. Nach jeder Rotation prüfen wir also, ob wir diesen Array-Zustand schon einmal hatten

oder nehmen ihn neu in unsere Sammlung auf. Sobald es einen Treffer (= eine Wiederholung) gibt, können wir uns die Differenz `diff` vom ersten Vorkommnis und dem jetzigen Durchlauf ausrechnen. Es ist klar, dass nun **immer** nach `diff` Rotationen wieder derselbe Zustand vorgefunden werden würde. Wir können nun also direkt so häufig diff auf den aktuellen Durchlauf drauf addieren, solange der Wiederholungs-Counter kleiner 1 Milliarde ist. Nur die letzten paar Rotationen müssen nochmals explizit durchgeführt werden.

- Zum Speichern dieser Array-Zustände und der Zahl der Rotation nutzen wir eine HashMap. Diese muss pro Eintrag immer aus zwei Werten bestehen: einem Key und einem Value. Der Key beschreibt den eindeutigen Array-Zustand und der Value ist die Zahl der Rotation. Um den Array-Zustand eindeutig zu beschreiben bietet es sich z. B. an, einen String zu erstellen der aus den zeilenweise gelesenen Abständen aller 0's zueinander besteht. Eine Zeile mit `OO..O.##O` würde Z. B. zu `0023` werden.

Lösungsvorschlag

```
public long partTwo() {
    int summe = 0;
    HashMap<String, Integer> cache = new HashMap<>();

    // Lies alles in ein char[][] ein
    char[][] input = new
char[inputLines.get(0).length()][inputLines.size()];
    for (int y = 0; y < inputLines.size(); y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < line.length(); x++) {
            input[x][y] = line.charAt(x);
        }
    }

    int breite = inputLines.get(0).length();
    int hoehe = inputLines.size();
    // 1 Millarde mal drehen
    int i = 0;
    while (i < 1_000_000_000) {
        input = moveUp(input, hoehe, breite);
        input = moveLeft(input, hoehe, breite);
        input = moveDown(input, hoehe, breite);
        input = moveRight(input, hoehe, breite);
        i++;

        // berechne key-String (speichere dir alle Abstände der 0er
zueinander (zeilenweise))
        String key = "";
        int counter = 0;
        for (int y = 0; y < hoehe; y++) {
            for (int x = 0; x < breite; x++) {
                if (input[x][y] == 'O') {
                    key += counter;
                    counter = 0;
                } else {
```

```
                    counter++;
                }
            }
        }

        if (cache.containsKey(key)) {
            int old = cache.get(key);
            int diff = i - old;
            while (i+diff < 1_000_000_000) {
                i += diff;
            }
        } else {
            cache.put(key, i);
        }
    }


    for (int y = 0; y < inputLines.size(); y++) {
        for (int x = 0; x < inputLines.get(0).length(); x++) {
            if (input[x][y] == 'O') {
                summe += inputLines.size() - y;
            }
        }
    }


    return summe;
}
```