Day 11: Plutonian Pebbles

Dieser Tag ist etwas kniffliger. Teil 1 kann man noch zur **mittleren Schwierigkeit** zählen, Teil 2 hingegen zählt definitiv zur **höchsten Schwierigkeitsstufe**, da man **Memoisation bzw. dynamische Programmierung** benötigt. Das bedeutet, dass man Zwischenergebnisse speichern muss, um bei erneuten rekursiven Aufrufen mit denselben Parametern direkt das vorherige Ergebnis zurückliefern zu können. Andernfalls würde die Rechenzeit für Teil 2 explodieren. Hierzu wird eine **HashMap** benötigt.

Teil 1

Teil 1 lässt sich noch relativ *straight-forward* lösen. Man speichert die Input-Zahlen also in einer ArrayList und iteriert 25x über die Schleife. Pro Iteration erstellt man am besten eine **neue** ArrayList, die man mit den neuen Werten füllt, die sich nach den Regeln aus der **alten** ArrayList (des letzten Durchgangs) ergeben.

Lösungsvorschlag

```
public void partOne() {
    // Übertrage die Input-String-Zahlen in ein Array
    ArrayList<Long> input = new ArrayList();
    for (String n: inputLines.get(0).split(" ")) {
        input.add(Long.parseLong(n));
    }

    // 25x "blinzeln"
    for (int blink = 0; blink < 25; blink++) {
        // erstelle pro Durchgang eine neue ArrayList für die neuen Werte
        ArrayList<Long> next = new ArrayList();

        // pro altem Wert des letzten Durchgangs
        for (long i: input) {
            if (i == 0) {
                next.add((long)1);
            } else if (String.valueOf(i).length() % 2 == 0) {
                // Aufteilung in vordere/hintere Hälfte der Zahl
                next.add((long) (i /
Math.pow(10,(String.valueOf(i).length()/2))));
                next.add((long) (i %
Math.pow(10,(String.valueOf(i).length()/2))));
            } else {
                next.add(i * 2024);
            }
        }
        // mache die aktuelle/neue ArrayList "next" zur alten für den
nächsten Durchlauf
        input = next;
```

```
    }
    System.out.println(input.size());
}
```

Teil 2

Eine ausführliche Erklärung würde hier vermutlich den Rahmen sprengen. Der Quellcode des Lösungsvorschlags ist ein wenig kommentiert...

Lösungsvorschlag

```
public void partTwo() {
    // Speichert pro "Steinzahl" ein Array, das wiederum pro Zeitschritt
0-74 die bereits
    // errechneten Ergebnisse (Anzahl der Steine bei Zeitschritt 75)
speichert.
    memo = new HashMap();

    // Übertrage die Input-String-Zahlen in ein Array
    ArrayList<Long> input = new ArrayList();
    for (String n: inputLines.get(0).split(" ")) {
        input.add(Long.parseLong(n));
    }

    // Starte pro Input-Zahl den rekursiven Aufruf
    long stones = 0;
    for (long i: input) {
        stones += calculate(i, 0);
    }
    System.out.println(stones);
}

private long calculate(long i, int times) {
    // Letzter Zeitschritt? Dann gibts genau einen (weiteren) Stein
    if (times == 75) {
        return 1;
    }

    // Schaue im Speicher der Zwischenergebnisse nach, ob schon einmal die
Anzahl der Steine
    // beim Input i zum Zeitpunkt times berechnet wurde.
    long[] stones = memo.get(i);
    if (stones != null && stones[times] != 0)  {
        return stones[times];
    } else if (stones == null) {
        // Wenn zu dieser Zahl noch nie nachgeschaut wurde, dann erstelle
schon mal das leere Array für die verschiedenen Zeitpunkte
```

```
        stones = new long[75];
    }

    long result = 0;
    if (i == 0) {
        result += calculate((long)1,times+1);
    } else if (String.valueOf(i).length() % 2 == 0) {
        result += calculate((long) (i /
Math.pow(10,(String.valueOf(i).length()/2))), times+1);
        result += calculate((long) (i %
Math.pow(10,(String.valueOf(i).length()/2))), times+1);
    } else {
        result += calculate(i * 2024,times+1);
    }

    // Speichere das neue Ergebnis für den Zeitpunkt times
    stones[times] = result;
    // speichere das ganze Array der verschiedenen Zeitpunkte in der HashMap
passend zum Wert i
    memo.put(i, stones);

    return result;
}
```