# Day 22: Monkey Market

Teil 1 ist **sehr einfach** zu implementieren! Für Teil 2 hingegen benötigt man mindestens eine HashMap und die "Fähigkeit die Übersicht zu bewahren".

## Teil 1

Lest zunächst genau die Aufgabenbeschreibung - insbesondere, was pro Sequenz alles gerechnet werden muss. Die dort genannten Rechnungen kann man exakt in der Form und in der Reihenfolge implementieren. Achtung, die Zahlen werden teils sehr groß, nehmt unbedingt überall `long` statt `int`. Zum Einlesen der Eingabewerte entsprechend: `Long.parseLong(…)`.

In Kurzform: Pro Zeile musst du in einer Schleife 2000x die genannten Sequenz-Berechnungen durchführen. Das Endergebnis summierst du über alle Zeilen auf.

Bitweises XOR kannst du mit ^ erreichen. Beispiel: `secret = secret ^ result`, wenn `result` das Ergebnis der vorherigen Rechnung (wie z. B. `secret * 64`) enthält.

Lösungsvorschlag

```
public void partOne() {
    long result = 0;
    for (String line: inputLines) {
        long secret = Long.parseLong(line);
        for (int i = 0; i < 2000; i++) {
            long res = secret * 64;
            secret = secret ^ res;
            secret = secret % 16777216;
            res = secret / 32;
            secret = secret ^ res;
            secret = secret % 16777216;
            res = secret * 2048;
            secret = secret ^ res;
            secret = secret % 16777216;
        }
        result += secret;
    }

    System.out.println(result);
}
```

## Teil 2

Für Teil 2 gibt es hier nur den Lösungsvorschlag.

Lösungsvorschlag

```
public void partTwo() {
    long result = 0;

    // Speichert pro "4 changes" die Summe der damit erreichbaren Preise
    HashMap<List<Integer>, Integer> hashmap = new HashMap();
    for (String line: inputLines) {
        long secret = Long.parseLong(line);

        // Speichert, ob ein bestimmtes Array von "4 changes" für diese
Eingabe bereits gefunden wurde
        HashMap<List<Integer>, Boolean> alreadyFound = new HashMap();

        // Array, um die letzten 4 Changes zu speichern.
        Integer[] changes = new Integer[4];
        for (int i = 0; i < 4; i++) {
            changes[i] = 0;
        }

        // speichert die nächste Position, um im Array den nächsten Wert
einzufügen.
        // datei "rotiert" changepos dank dem Modulo-Operator immer zwischen
den Indizes 0-3!
        int changePos = 0;

        int previousPrice = 0;
        int price = (int)secret % 10; // letzte Ziffer des Secrets
        for (int i = 0; i < 2000; i++) {
            // Alle Berechnungen einer Sequenz
            long res = secret * 64;
            secret = secret ^ res;
            secret = secret % 16777216;
            res = secret / 32;
            secret = secret ^ res;
            secret = secret % 16777216;
            res = secret * 2048;
            secret = secret ^ res;
            secret = secret % 16777216;

            previousPrice = price;
            price = (int)secret % 10;
            changes[changePos] = price - previousPrice;
            changePos++;
            changePos %= 4;

            // "sortiert" die letzten 4 changes in die korrekte Reihenfolge
            Integer[] changesOrdered = order(changes, changePos);
            // wenn genügend changes vorhanden sind UND wenn die Folge noch
```

```
NICHT gefunden wurde...
            if (i >= 3 && alreadyFound.get(Arrays.asList(changesOrdered)) ==
null) {
                // Ändere die Summe der Preise für die spezifische
Folge/Liste
                hashmap.put(Arrays.asList(changesOrdered),
hashmap.getOrDefault(Arrays.asList(changesOrdered),0) + price);
                alreadyFound.put(Arrays.asList(changesOrdered), new
Boolean(true));
            }
        }
        result += secret;
    }

    System.out.println(Collections.max(hashmap.entrySet(),
Map.Entry.comparingByValue()).getValue());
}

private Integer[] order(Integer[] changes, int changePos) {
    Integer[] res = new Integer[4];
    for (int i = 0; i < 4; i++) {
        res[i] = new Integer(changes[changePos]);
        changePos++;
        changePos %= 4;
    }
    return res;
}
```