Day 5: Print Queue

Teil 1 ist vom Verständnis noch recht gut zu begreifen, beim Programmieren muss man aber trotzdem bereits etwas um die Ecke denken. Bei Teil 2 muss man dann erst recht einen Kniff erkennen, um sich das Leben einfacher zu machen. Daher zählt der Tag zur mittleren Schwierigkeit.

Teil 1

Tipps zur Vorgehensweise:

- Speichere alle Rules in einer ArrayList<int[]>. Wandle also jede Rule in ein zweistelliges Array um.
- Speichere alle Updates in einer ArrayList<int[]>. Wandle also jedes Update ebenso in ein Array um (so lang, wie es jeweils nötig ist).
- Prüfe dann für jedes Update, ob eine Rule dabei ist, die nicht zum aktuellen Update passt.
    - Überprüfe dazu pro Rule am besten, an welchem Index innerhalb des Updates die erste Zahl der Rule ist und an welchem Index die zweite Zahl ist.
    - Wenn anschließend der Index der ersten Zahl **größer** ist als der Index der zweiten Zahl, dann muss das Update falsch sein.

Lösungsvorschlag

```
public void partOne() {
    // beide als Instanzvariable hinterlegt (für Teil 2 am sinnvollsten)
    rules = new ArrayList();
    updates = new ArrayList();

    for (String line: inputLines) {
        // wenn eine rule erkannt wird...
        if (line.contains("|")) {
            String[] strRule = line.split("\\|"); // man muss den
senkrechten Strich "escapen"

            // speichere die rule in einem zweistelligen int-Array ab
            int[] intRule = new int[2];
            intRule[0] = Integer.parseInt(strRule[0]);
            intRule[1] = Integer.parseInt(strRule[1]);

            // füge dieses Array in die "Zusammenstellung" aller rules hinzu
            rules.add(intRule);


        } else if (line.contains(",")) {    // wenn ein update erkannt wird
            String[] strUpdate = line.split(",");

            // speichere die Update-Zeile als int-Array
            int[] intUpdate = new int[strUpdate.length];
```

```
            for (int i = 0; i < intUpdate.length; i++) {
                intUpdate[i] = Integer.parseInt(strUpdate[i]);
            }
            // füge das Array zur "Zusammenstellung" aller updates hinzu
            updates.add(intUpdate);
        }
    }

    // speichere darin die Summe aller mittleren Nummern der korrekten
Updates
    int middleNumbers = 0;

    // durchsuche alle updates
    for (int[] update: updates) {
        // gehe davon aus, dass das aktuelle Update in richtiger Ordnung
ist.
        boolean rightOrder = true;
        // Überprüfe nun jede Regel, ob auch nur eine verletzt wird!
        for (int[] rule: rules) {
            // Suche nun für die linke Zahl der Regel den Index innerhalb
der Update-Zeile
            // ... und das selbe für die rechte Zahl der Regel.
            // wenn eine Zahl nicht im Update enthalten ist, dann wird -1
als Position zurückgegeben
            int i1 = indexOf(update, rule[0]);
            int i2 = indexOf(update, rule[1]);
            // Abbruchbedingung, wenn beide Zahlen im Update enthalten
sind... (Regel ist für das Update wichtig)
            // ... UND wenn der Index der Zahlen in der Regel andersherum
angegeben ist!
            if (i1 != -1 && i2 != -1 && i1 > i2) {
                rightOrder = false;
                break; // Schleife frühzeitig abbrechen
            }
        }
        // nur, wenn die Zahlenanordnung im Update korrekt ist, dann wird
die mittlere Zahl addiert.
        if (rightOrder) {
            middleNumbers += update[update.length/2];
        }
    }

    System.out.println(middleNumbers);
}

/**
 * Gibt den Index zurück, an dem "element" in "array" gefunden wird.
 * Gibt -1 zurück, wenn das Element nicht gefunden wird
 * @param array zu durchsuchendes Array
```

```
 * @param element gesuchtes Element
 * @return Index oder -1, wenn nicht vorhanden
 */
private int indexOf(int[] array, int element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return i;
        }
    }
    return -1;
}
```

Teil 2

Teil 2 beginnt zunächst identisch zu Teil 1. Ebenso werden zwei ArrayLists angelegt, die alle Rules und alle Updates enthalten. Und ebenso wird dann jedes Update durchsucht, ob man ein inkorrektes Update gefunden hat. Wenn man ein solches gefunden hat, dann wird es aber knifflig.

- Man muss nun (zumindest gedanklich) eine Sortierung des inkorrekten Updates vornehmen, sodass alle Zahlen in der Sortierung vorliegen, wie sie sich nach den Regeln ergibt. Eine tatsächliche Sortierung vorzunehmen wäre aber unheimlich komplex zu programmieren. Tatsächlich geht es einfacher!
- Anstatt die Updates tatsächlich zu sortieren suchen wir uns zunächst einmal nur diejenigen Regeln heraus, deren linke UND rechte Zahl im aktuellen Update vorkommt.
- Anschließend zählen wir für jede Zahl des Updates, wie häufig diese auf der linken und rechten Seite jeder relevanten Regel vorkommt. Diejenige Zahl, die in der Sortierung in der Mitte sein muss, die muss *genauso häufig rechts von einer anderen Zahl wie auch links von einer anderen Zahl* stehen. Wir suchen also diejenige Zahl, wo die linke und rechte Häufigkeit innerhalb der Regeln identisch ist.

Lösungsvorschlag

```
public void partTwo() {
    rules = new ArrayList();
    updates = new ArrayList();

    // siehe Teil 1!
    for (String line: inputLines) {
        if (line.contains("|")) {
            String[] strRule = line.split("\\|");
            int[] intRule = new int[2];
            intRule[0] = Integer.parseInt(strRule[0]);
            intRule[1] = Integer.parseInt(strRule[1]);
            rules.add(intRule);
        } else if (line.contains(",")) {
            String[] strUpdate = line.split(",");
            int[] intUpdate = new int[strUpdate.length];
            for (int i = 0; i < intUpdate.length; i++) {
                intUpdate[i] = Integer.parseInt(strUpdate[i]);
```

```
            }
            updates.add(intUpdate);
        }
    }

    int middleNumbers = 0;
    // ähnlich wie bei Teil 1: Durchsuche jedes Update und finde die
ungültigen Updates
    for (int[] update: updates) {
        for (int[] rule: rules) {
            int i1 = indexOf(update, rule[0]);
            int i2 = indexOf(update, rule[1]);
            // Finde ungültiges Update
            if (i1 != -1 && i2 != -1 && i1 > i2) {
                // berechne in einer separaten Methode die Mitte des
ungültigen Updates
                middleNumbers += findMiddleOfUpdate(update);
                break;
            }
        }
    }

    System.out.println(middleNumbers);
}

/**
 * @param update Das Update, das fehlerhaft geordnet ist
 * @return mittleres Element NACH der Sortierung
 */
private int findMiddleOfUpdate(int[] update) {
    // Neue ArrayList, um darin nur diejenigen Rules zu speichern, die für
das Update von Relevanz sind.
    ArrayList<int[]> rulesForUpdate = new ArrayList();

    // Durchsuche jede Rule und speichere nur diejenigen, bei denen die
linke UND rechte Zahl im Update vorkommt.
    for (int[] rule: rules) {
        if (indexOf(update, rule[0]) >= 0 && indexOf(update, rule[1]) >= 0)
{
            rulesForUpdate.add(rule);
        }
    }

    // Zähle, wie häufig jede Zahl des Updates an der linken oder rechten
Stelle in den Regeln vorkommt.
    for (int zahl: update) {
        int links = 0;
        int rechts = 0;
        for (int[] rule: rulesForUpdate) {
```

```
            if (rule[0] == zahl) {
                links++;
            }
            if (rule[1] == zahl) {
                rechts++;
            }
        }
        if (links == rechts) {
            return zahl;
        }
    }
    return -1;
}
```