Tag 19 - Aplenty

- Variante 1

Lösungshinweise Teil 1

- Speichere jeden Workflow in einer Hashmap von <String, Workflow>. Nutze für jeden Workflow (jede Zeile) eine eigene Klasse.
- Der Key (String) ist dabei der vorderste String des Workflows vor den geschweiften Klammern.
- Die Klasse **Workflow** speichert eine ArrayList von String-Arrays. Jede einzelne Regel wird in ihre 4 Bestandteile zerlegt (x/m/a/s, größer/kleiner, Vergleichswert und Folgezustand). Diese 4 Bestandteile werden in einem String-Array gespeichert. Da die Buchstaben x/m/a/s immer in dieser Reihenfolge bei den Parts (letzte Zeilen des Inputs) erscheinen, ist es äußerst sinnvoll, diese direkt durch die Indizes 0, 1, 2 und 3 zu ersetzen. Der letzte Zeil des Workflows (der sonst-Fall) kann separat als String gespeichert werden.
- Außerdem benötigt die Klasse **Workflow** noch eine Methode `calculate(int[] in)`, welche ein int[]-Array als Parameter entgegennimmt (dazu gleich), und den Nachfolge-Zustand zurückgibt: Überprüfe für jede einzelne Regel im Workflow, ob ein Vergleich `true` ergibt, dann gib den Nachfolge-Zustand zurück.
- In der Hauptmethode müssen nach den Workflows noch die Teile/Parts eingelesen werden. Speichere jeweils die 4 Zahlenwerte direkt in einem int[]-Array der Größe 4 hab und stecke alle Arrays in eine ArrayList<int[]>.
- Prüfe nun für jeden Part in einer Endlosschleife immer wieder was der Nachfolge-Workflow ist. Sobald der Workflow "A" oder "R" ist, kannst du mit `break` die Schleife abbrechen.

Lösungsvorschlag Klasse Workflow

```
import java.util.ArrayList;

public class Workflow
{
    private ArrayList<String[]> steps;
    private String otherwise;

    public Workflow(String s)
    {
        steps = new ArrayList<String[]>();

        String[] sections = s.split(",");
        for (int i = 0; i < sections.length - 1; i++) {
            String[] section = new String[4];

            String ch = sections[i].split("[<>]")[0];
            if (ch.equals("x")) {
                section[0] = "0";
            } else if (ch.equals("m")) {
                section[0] = "1";
            } else if (ch.equals("a")) {
```

```
                section[0] = "2";
            } else {
                section[0] = "3";
            }

            if (sections[i].contains("<")) {
                section[1] = "<";
            } else {
                section[1] = ">";
            }

            section[2] = sections[i].split("[<>]")[1].split(":")[0];
            section[3] = sections[i].split(":")[1];
            steps.add(section);
        }
        otherwise = sections[sections.length-1];
    }

    public String calculate(int[] in) {
        for (String[] step: steps) {
            if (step[1].equals("<") && in[Integer.parseInt(step[0])] <
Integer.parseInt(step[2])) {
                return step[3];
            } else if (step[1].equals(">") && in[Integer.parseInt(step[0])]
> Integer.parseInt(step[2])) {
                return step[3];
            }
        }
        return otherwise;
    }
}
```

Lösungsvorschlag Teil 1

```
private HashMap<String, Workflow> workflow;
private ArrayList<int[]> parts;

public int partOne() {
    workflow = new HashMap<String, Workflow>();
    parts = new ArrayList<int[]>();

    for (String line: inputLines) {
        if (line.length() == 0) {
            continue;
        } else if (!line.startsWith("{")) {
            workflow.put(line.split("\\{")[0], new
Workflow(line.split("\\{")[1].split("\\}")[0]));
        } else {
```

```
            int[] part = new int[4];
            String[] xmas = line.split("\\{")[1].split("\\}")[0].split(",");
            for (int i = 0; i < 4; i++) {
                part[i] = Integer.parseInt(xmas[i].split("=")[1]);
            }
            parts.add(part);
        }
    }

    int summe = 0;
    for (int[] part: parts) {
        String nextKey = "in";
        while(true) {
            nextKey = workflow.get(nextKey).calculate(part);
            if (nextKey.equals("R")) {
                break;
            } else if (nextKey.equals("A")) {
                for (int p: part) {
                    summe += p;
                }
                break;
            }
        }
    }

    return summe;
}
```

Lösungsvorschlag Teil 2

- Gehe rückwärts vor: suche alle A(accepted) in den Workflows/Regeln und gehe von dort aus die Regeln rückwärts durch, bis du bei der Regel "in" gelandet bist. Merke dir auf dem Weg dahin alle größer/kleiner-Vergleiche. Setze anhand dieser Vergleiche die Maximums- und Minimumsgrenzen, um die erlaubte Differenz pro x/m/a/s zu bilden.

Lösungsvorschlag Teil 2

```
private ArrayList<ArrayList<String[]>> allPaths;

public long partTwo() {
    workflow = new HashMap<String, Workflow>();

    // lies nur die workflows ein
    for (String line: inputLines) {
        if (line.length() == 0) {
            continue;
        } else if (!line.startsWith("{")) {
            workflow.put(line.split("\\{")[0], new
Workflow(line.split("\\{")[1].split("\\}")[0]));
```

```
        }
    }

    Workflow start = workflow.get("in");
    allPaths = new ArrayList();
    ArrayList<String[]> path = new ArrayList<String[]>();
    getPathsToA(start, 0, path);

    long ergebnis = 0;
    for (ArrayList<String[]> p: allPaths) {
        int[][] range = new int[4][2];
        for (int i = 0; i < 4; i++) {
            range[i][0] = 1;
            range[i][1] = 4000;
        }
        for (String[] r: p) {
            int comp;
            if (r[1].equals(">") && Integer.parseInt(r[2]) >
range[Integer.parseInt(r[0])][0]) {
                range[Integer.parseInt(r[0])][0] = Integer.parseInt(r[2]);
            } else if (r[1].equals("<") && Integer.parseInt(r[2]) <
range[Integer.parseInt(r[0])][1]) {
                range[Integer.parseInt(r[0])][1] = Integer.parseInt(r[2]);
            }
        }

        long teilergebnis = 1;
        for (int i = 0; i < 4; i++) {
            teilergebnis *= range[i][1] - range[i][0] + 1;
        }
        ergebnis += teilergebnis;
    }

    return ergebnis;
}

private void getPathsToA(Workflow w, int step, ArrayList<String[]> path) {

    if (step >= w.getSteps().size()) {
        String nachfolger = w.getOtherwise();
        if (nachfolger.equals("A")) {
            allPaths.add(path);
        } else if (nachfolger.equals("R")) {
            // do nothing
        } else {
            getPathsToA(workflow.get(nachfolger), 0,
(ArrayList<String[]>)path.clone());
        }
        return;
```

```
    }

    String[] stepX = w.getSteps().get(step);

    // annahme, es sei false -> gehe zum nächsten step
    ArrayList<String[]> p1 = (ArrayList<String[]>)path.clone();
    //invertieren
    if (stepX[1].equals("<")) {
        p1.add(new String[]{stepX[0], ">", stepX[2]});
    } else if (stepX[1].equals(">")) {
        p1.add(new String[]{stepX[0], "<", stepX[2]});
    }
    getPathsToA(w, step + 1, p1);

    // annahme, es sei true -> gehe zum neuen workflow
    if (stepX[1].equals("<")) {
        path.add(new String[]{stepX[0], "<",
String.valueOf(Integer.parseInt(stepX[2])-1)});
    } else if (stepX[1].equals(">")) {
        path.add(new String[]{stepX[0], ">",
String.valueOf(Integer.parseInt(stepX[2])+1)});
    }
    if (stepX[3].equals("A")) {
        allPaths.add(path);
        return;
    } else if (stepX[3].equals("R")) {
        return;
    }
    getPathsToA(workflow.get(stepX[3]), 0, path);

}
```