Tag 20 - Pulse Propagation

- Variante 1

Lösungshinweise Teil 1

- Die Pulse können mit einer eigenen Klasse dargestellt werden. Jeder Puls hat einen Vorgänger, einen Nachfolger und einen Zustand (z. B. 1/0 für high/low).

Lösungsvorschlag Pulse

```
public class Pulse
{
    private String from, to;
    private int pulse;

    public Pulse(String from, String to, int pulse)
    {
        this.from = from;
        this.to = to;
        this.pulse = pulse;
    }

    public String getFrom() {
        return this.from;
    }

    public String getTo() {
        return this.to;
    }

    public int getPulse() {
        return this.pulse;
    }
}
```

- Da verschiedene Pulse-Module vorkommen (insbesondere Flip-Flops und Conjunctions), die intern unterschiedlich funktionieren aber auf gleiche Weise genutzt werden, bietet sich eine OOP-Lösung inklusive Vererbung an. Speichere jede Zeile als in `PulseModule` und speichere alle PulseModules in einer HashMap.
- Die Oberklasse `PulseModule` ist abstrakt. Jede Unterklasse muss ihren eigenen Namen kennen und die Namen aller Nachfolger-Module. Außerdem muss jede Unterklasse die Methode `pulse()` besitzen. Diese berechnet, welche Ausgabepulse bei einem gegebenen Eingabepuls entstehen.

Lösungsvorschlag PulseModule

```
import java.util.ArrayList;

public abstract class PulseModule
{
    public abstract Pulse[] pulse(Pulse in);
    public abstract ArrayList<String> getNachfolger();
    public abstract String getModuleName();
}
```

- Die erste Unterklasse `FlipFlop` muss den aktuellen Zustand speichern können und ansonsten nur die `pulse()`-Methode implementieren.

Lösungsvorschlag FlipFlop

```
import java.util.ArrayList;

public class FlipFlop extends PulseModule
{
    private boolean state; // on or off
    private ArrayList<String> nachfolger;
    private String myName;

    public FlipFlop(String myName, ArrayList<String> nachfolger)
    {
        state = false;
        this.myName = myName;
        this.nachfolger = nachfolger;
    }

    public Pulse[] pulse(Pulse in) {
        if (in.getPulse() == 1) {
            return new Pulse[]{};
        } else {
            state = !state;

            Pulse[] pulses = new Pulse[nachfolger.size()];
            int p = 0;
            if (state) {
                p = 1;
            }

            for (int i = 0; i < nachfolger.size(); i++) {
                pulses[i] = new Pulse(myName, nachfolger.get(i), p);
            }
            return pulses;
        }
    }
```

```
    public String getModuleName() {
        return this.myName;
    }

    public ArrayList<String> getNachfolger() {
        return this.nachfolger;
    }
}
```

- Die zweite Unterklasse Conjunction muss als Besonderheit zu jedem der Vorgänger/Eingänge den letzten Zustand speichern (z. B. in einer HashMap). Die `pulse()`-Methode muss natürlich anders implementiert werden

Lösungsvorschlag Conjunction

```
import java.util.ArrayList;
import java.util.HashMap;

public class Conjunction extends PulseModule
{
    private HashMap<String,Integer> inputs = new HashMap<String,Integer>();
    private String myName;
    private ArrayList<String> nachfolger;

    public Conjunction(String myName, ArrayList<String> nachfolger)
    {
        this.myName = myName;
        this.nachfolger = nachfolger;
    }

    public void addInput(String input) {
        inputs.put(input, 0);
    }

    public Pulse[] pulse(Pulse in) {
        inputs.replace(in.getFrom(),in.getPulse());
        boolean high = true;
        for (Integer i: inputs.values()) {
            if (i == 0) {
                high = false;
                break;
            }
        }

        Pulse[] pulses = new Pulse[nachfolger.size()];
        int p = 1;
        if (high) {
            p = 0;
        }
```

```
        for (int i = 0; i < nachfolger.size(); i++) {
            pulses[i] = new Pulse(myName, nachfolger.get(i), p);
        }
        return pulses;
    }

    public String getModuleName() {
        return this.myName;
    }

    public ArrayList<String> getInputs() {
        ArrayList<String> keys = new ArrayList();
        for (String key: this.inputs.keySet()) {
            keys.add(key);
        }
        return keys;
    }

    public ArrayList<String> getNachfolger() {
        return this.nachfolger;
    }
}
```

- Der letzte Modul-Typ "Broadcaster" ist relativ langweilig, da er keine besondere Funktion hat.

Lösungsvorschlag Broadcaster

```
import java.util.ArrayList;

public class Broadcaster extends PulseModule
{
    private String myName;
    private ArrayList<String> nachfolger;

    /**
     * Konstruktor für Objekte der Klasse Broadcaster
     */
    public Broadcaster(String myName, ArrayList<String> nachfolger)
    {
        this.myName = myName;
        this.nachfolger = nachfolger;
    }

    public String getModuleName() {
        return this.myName;
    }

    public ArrayList<String> getNachfolger() {
```

```
        return this.nachfolger;
    }

    // dummy method
    public Pulse[] pulse(Pulse p) {
        return new Pulse[]{};
    }
}
```

- Erstelle aus jeder Eingabezeile die nötigen Puls-Module, die alle auch vom Typ der Oberklasse `PulseModule` sind. Daher kannst du alle in einer `HashMap<String,PulseModule>` speichern. Mit dieser HashMap kannst du für jeden Modulnamen (key) sofort das dazugehörige Modul bekommen, egal, ob es ein FlipFlop oder eine Conjunction ist.
- Du musst nach dem Erstellen der Module noch ein zweites Mal über jedes erstellte Modul iterieren, um die Vorgänger in die Conjunctions eintragen zu können (damit jede Conjunction Buch führen kann, ob ihre Vorgänger jeweils high oder low waren).
- Um einen einzelnen Durchlauf (Knopfdruck bis Ende) zu erreichen, und die Pulse in der richtigen Reihenfolge abzuarbeiten, musst du eine Queue nutzen. Stecke zu Beginn die Broadcast-Pulse in die Queue. Dann iteriere in einer while-Schleife solange die Schleife nicht leer ist:
  - nimm dir den nächsten Pulse aus der Queue
  - zähle, ob der Pulse high oder low ist
  - hole dir mithilfe des Ziels des Pulses das nächste Module aus der HashMap
  - rufe auf dem Module die `pulse()`-Methode auf und lass dir alle nachfolgenden Pulse zurückgeben
  - speichere jeden Nachfolge-Pulse in der Queue
- Tue diese letzten Schritt Knopfdruck-Durchlauf wiederum 1000 Mal

Lösungsvorschlag Teil 1

```
private HashMap<String, PulseModule> modules = new HashMap();

public long partOne() {
    String[] broadcasters = new String[0];
    for (String line: inputLines) {
        if (line.startsWith("broadcaster")) {
            String[] inits = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: inits) {
                s = s.trim();
            }
            broadcasters = inits;

            ArrayList<String> nachfolger = new ArrayList<String>();
            nachfolger.addAll(Arrays.asList(inits));

            modules.put("broadcaster", new Broadcaster("broadcaster",
nachfolger));
        } else if (line.startsWith("%")) {
            String moduleName = line.split("-")[0].split("%")[1].trim();
```

```
            ArrayList<String> nachfolger = new ArrayList();
            String[] nf = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: nf) {
                nachfolger.add(s.trim());
            }

            modules.put(moduleName, new FlipFlop(moduleName, nachfolger));
        } else if (line.startsWith("&")) {
            String moduleName = line.split("-")[0].split("&")[1].trim();

            ArrayList<String> nachfolger = new ArrayList();
            String[] nf = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: nf) {
                nachfolger.add(s.trim());
            }

            modules.put(moduleName, new Conjunction(moduleName,
nachfolger));
        }
    }

    // gehe alle module durch, um die vorgänger der conjuctions einzutragen
    for (PulseModule m: modules.values()) {
        ArrayList<String> nachfolger = m.getNachfolger();
        for (String n: nachfolger) {
            PulseModule nachfolgeModule = modules.get(n);
            if (nachfolgeModule instanceof Conjunction) {
                ((Conjunction)nachfolgeModule).addInput(m.getModuleName());
            }
        }
    }


    long lowPulses = 1000;
    long highPulses = 0;

    for (int i = 0; i < 1000; i++) {
        // Fülle die Queue mit den ersten Pulses vom Broadcaster
        Queue<Pulse> queue = new LinkedList<>();
        for (String s: broadcasters) {
            queue.offer(new Pulse("broadcaster", s, 0));
        }

        while(queue.size() > 0) {
            Pulse p = queue.poll();

            if (p.getPulse() == 0) {
```

```
                lowPulses++;
            } else {
                highPulses++;
            }

            PulseModule pm = modules.get(p.getTo());
            if (pm == null) {
                continue;
            }

            Pulse[] nachfolger = pm.pulse(p);
            for (Pulse nfp: nachfolger) {
                queue.offer(nfp);
            }
        }
    }

    return lowPulses*highPulses;
}
```

Lösungshinweise Teil 2

- Für Teil 2 ist es wichtig, den folgenden Kniff zu sehen: Der Output ist low, wenn **alle** Eingänge der letzten Conjunction high sind. Dies passiert aber erst nach vielen Billionen (!) Iterationen. Daher muss man noch einen Schritt weiter denken: Schaue dir jeden Vorgänger der letzten Conjunction an und speichere dir, wann diese Module jeweils **einzeln** einen high-Pulse an die letzte Conjunction aussenden! Multipliziere am Ende alle diese minimalen Iterationen für die jeweiligen high-Pulses.

Lösungsvorschlag Teil 2

```
public long partTwo() {
    String[] broadcasters = new String[0];
    for (String line: inputLines) {
        if (line.startsWith("broadcaster")) {
            String[] inits = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: inits) {
                s = s.trim();
            }
            broadcasters = inits;

            ArrayList<String> nachfolger = new ArrayList<String>();
            nachfolger.addAll(Arrays.asList(inits));

            modules.put("broadcaster", new Broadcaster("broadcaster",
nachfolger));
        } else if (line.startsWith("%")) {
            String moduleName = line.split("-")[0].split("%")[1].trim();
```

```
            ArrayList<String> nachfolger = new ArrayList();
            String[] nf = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: nf) {
                nachfolger.add(s.trim());
            }

            modules.put(moduleName, new FlipFlop(moduleName, nachfolger));
        } else if (line.startsWith("&")) {
            String moduleName = line.split("-")[0].split("&")[1].trim();

            ArrayList<String> nachfolger = new ArrayList();
            String[] nf = line.split(">")[1].replaceAll("\\s+",
"").split(",");
            for (String s: nf) {
                nachfolger.add(s.trim());
            }

            modules.put(moduleName, new Conjunction(moduleName,
nachfolger));
        }
    }

    // gehe alle module durch, um die vorgänger der conjuctions einzutragen
    for (PulseModule m: modules.values()) {
        ArrayList<String> nachfolger = m.getNachfolger();
        for (String n: nachfolger) {
            PulseModule nachfolgeModule = modules.get(n);
            if (nachfolgeModule instanceof Conjunction) {
                ((Conjunction)nachfolgeModule).addInput(m.getModuleName());
            }
        }
    }

    // finde letzte conjunction
    String letzteConjunction = "";
    for (PulseModule m: modules.values()) {
        ArrayList<String> nachfolger = m.getNachfolger();
        for (String n: nachfolger) {
            if (modules.get(n) == null) {
                letzteConjunction = m.getModuleName();
            }
        }
    }

    // finde alle Vorgänger dieser letzten Conjunction
    HashMap<String,Integer> vorletzteConjunctions = new HashMap();
    for (String s:
```

```
((Conjunction)modules.get(letzteConjunction)).getInputs()) {
        vorletzteConjunctions.put(s, 0);
    }

    long i = 0;
    //HashMap<String,Integer> statesFound = new HashMap<String,Integer>();

    while (true) {
        i++;

        long result = 1;
        for (Integer iteration: vorletzteConjunctions.values()) {
            result *= iteration;
        }
        if (result != 0) {
            System.out.println(result);
            return result;
        }

        // Fülle die Queue mit den ersten Pulses vom Broadcaster
        Queue<Pulse> queue = new LinkedList<>();
        for (String s: broadcasters) {
            queue.offer(new Pulse("broadcaster", s, 0));
        }

        while(queue.size() > 0) {
            Pulse p = queue.poll();

            PulseModule pm = modules.get(p.getTo());
            if (pm == null) {
                if (p.getPulse() == 0) {
                    return i;
                } else {
                    continue;
                }
            }

            Pulse[] nachfolger = pm.pulse(p);
            for (Pulse nfp: nachfolger) {
                // wenn eine vorletzteConjunction HIGH sendet, dann merken!
                if (vorletzteConjunctions.get(nfp.getFrom()) != null &&
nfp.getPulse() == 1) {
                    vorletzteConjunctions.put(nfp.getFrom(), (int)i);
                }

                queue.offer(nfp);
            }
        }
    }
}
```