Tag 4

- Aufgabe
- Inputdateien

d4e.txt, Input d4i.txt.

Kontrollergebnisse

- Eingabedatei d4e.txt: [Teil 1 - 13] [Teil 2 - 30]
- Eingabedatei d4i.txt: [Teil 1 - 26443] [Teil 2 - 6284877]

- Variante 1
- Variante 2 - Objektorientiert

Hilfestellungen Teil 1

- Grundidee: Nutze die Methode `split("…")`, um die Zeile so zu unterteilen, dass du zunächst alle Zahlen vor unter hinter dem Trennzeichen | bekommst. Nutze es erneut, um vom vorderen Teil wiederum nur die Zahlen hinter dem Doppelpunkt zu bekommen. Splitte die beiden Strings mit Zahlen erneut an den Leerzeichen, um alle einzelnen Zahlen als Stringarray zu bekommen, diese kannst du dann der Reihe nach mit `Integer.parse(…)` in int-Zahlen umwandeln. Anschließend überprüfst du für jede hintere Zahl, ob sie in den vorderen Zahlen vorkommt.
- Bei zwei Verwendungen von `split` musst du aufpassen! **Erstens**: Der senkrechte Strich wird im Java-String als Metazeichen erkannt und muss "escaped" werden (also die versteckte Funktion/Bedeutung muss entfernt werden) `\\|`. **Zweitens**: zwischen manchen Zahlen sind mehrere Leerzeichen. Du musst also machnmal mehrere Leerzeichen zum Splitten nutzen. Das kannst du mit dem "regulären Ausdruck" `\\s+` machen, dieser erkennt beliebig viele "Whitespaces" (also u.a. das Leerzeichen).
- Zähle, wie häufig die hinteren Zahlen in den vorderen Zahlen vorkamen. Die Punkte pro Zeile berechnest du dann mit 2^(anzahl-1)

Lösungsvorschlag 1

```
private int[] stringToIntArray(String str) {
    String[] numbersStr = str.trim().split("\\s+");
    int[] array = new int[numbersStr.length];
    for (int i = 0; i < numbersStr.length; i++) {
        array[i] = Integer.parseInt(numbersStr[i].trim());
    }
    return array;
}

public int partOne() {
    int summe = 0;

    for (String line: inputLines) {
        String winningNumbersStr, numbersYouHaveStr;
        winningNumbersStr = line.split("\\|")[0].trim().split(":")[1];
```

```
        numbersYouHaveStr = line.split("\\|")[1].trim();

        int[] winningNumbers, numbersYouHave;
        winningNumbers = stringToIntArray(winningNumbersStr);
        numbersYouHave = stringToIntArray(numbersYouHaveStr);

        int matches = 0;
        for (int number : numbersYouHave) {
            for (int winningNumber : winningNumbers) {
                if (number == winningNumber) {
                    matches++;
                }
            }
        }
        summe += Math.pow(2, matches-1);
    }

    return summe;
}
```

Teil 2

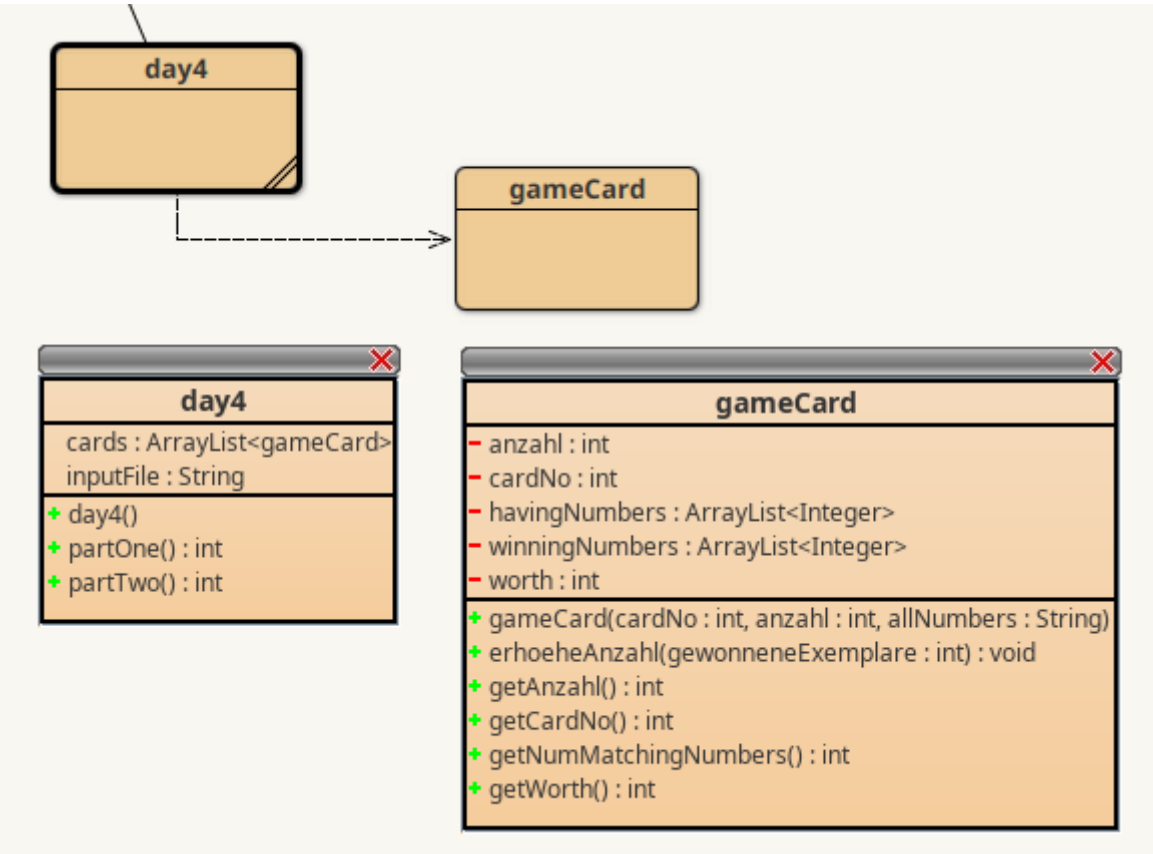
Aus Zeitgründen hier nur ein Lösungsvorschlag (Methoden aus Teil 1 werden benötigt):

Lösungsvorschlag

```
public int partTwo() {
    int summe = 0;

    int[] amountOfCards = new int[inputLines.size()];
    for (int i = 0; i < amountOfCards.length; i++) {
        amountOfCards[i] = 1;
    }

    for (int l = 0; l < inputLines.size(); l++) {
        String line = inputLines.get(l);

        String winningNumbersStr =
line.split("\\|")[0].trim().split(":")[1];
        String numbersYouHaveStr = line.split("\\|")[1].trim();

        int[] winningNumbers, numbersYouHave;
        winningNumbers = stringToIntArray(winningNumbersStr);
        numbersYouHave = stringToIntArray(numbersYouHaveStr);

        int numberOfMatches = 0;
        for (int number : numbersYouHave) {
            for (int winningNumber : winningNumbers) {
```

```
                if (number == winningNumber) {
                    numberOfMatches++;
                }
            }
        }

        for (int i = l+1; numberOfMatches > 0; i++) {
            amountOfCards[i] += amountOfCards[l];
            numberOfMatches--;
        }
    }

    for (int i : amountOfCards) {
        summe += i;
    }
    return summe;
}
```

Die grundsätzlichen Tipps aus Variante 1 haben Bestand - allerdings wird zunächst eine ArrayList des Typs `gameCards` befüllt, die dann mit entsprechenden Methoden alle Infos liefern, die man zur Lösung des Rätsels benötigt. Anstelle des regulären Ausdrucks für mehrere Leerzeichen wird in Variante 2 überprüft, ob ein String leer ist, und gegebenenfalls verworfen.



Das Spielkartenobjekt hat die Attribute:

- cardNo - Kartennummer
- worth - Der Wert der Karte gemäß des Rätsels Teil 1.
- anzahl - Die Anzahl, die man von dieser Karte besitzt. In Teil 1 nicht nötig.

- winningNumbers - ArrayList mit den Gewinnzahlen.
- havingNumbers - ArrayList mit den Zahlen,die man "hat".

Die Klasse day4 sieht dann ungefähr so aus:

```
 [...]
 ArrayList<gameCard> cards;
 [...]
 public day4() throws Exception {
    // Lese die Eingabedatei
    this.readInput(inputFile);

    cards = new ArrayList<>();
    for(String line: inputLines) {
        int cardNum =
Integer.parseInt(line.split(":")[0].split("Card")[1].strip());
        String numberString = line.split(":")[1].strip();
        cards.add(new gameCard(cardNum, 1, numberString));
    }
}
[...]
```

Der Konstruktor der Klasse gameCard erledigt die Hauptarbeit:

```
[...]
private ArrayList<Integer> winningNumbers;
private ArrayList<Integer> havingNumbers;
private int cardNo;
private int worth;
private int anzahl;

[...]

public gameCard(int cardNo, int anzahl, String allNumbers)
    {
        winningNumbers = new ArrayList<>();
        havingNumbers = new ArrayList<>();

        this.cardNo = cardNo;
        this.anzahl = anzahl;

        // FIXME: Befülle die Arraylists winningNumbers und havingNumbers
        // indem du den String allNumbers am | splittest, die beiden Teile
        // dann an den Leerzeichen aufteilst, in Integers umwandelst und in
die
        // ArrayLists einfügst.


        // FIXME Implementiere die Methode calculateWorth, die
```

```
        // eine ganze Zahl zurückliefert und den Wert für die
        // Karte hier korrekt setzt.
        this.worth = calculateWorth();

    }

    // Das geht ohne die Potenzfunktion, wenn man genau nachdenkt!
    private int calculateWorth() {
        int worth = 0;
        //FIXME

        return worth;
    }
```

Hinweise Teil 1

Nach dieser Vorbereitung ist Teil 1 trivial: Man iteriert in einer Schleife durch alle Karten und addiert die Kartewerte, die man sich mit einem Getter `getWorth()` liefern lassen kann. Berechnet wurden die Werte bereits beim anlegen der Kartenliste.

Hinweise Teil 2

- Für Teil 2 benötigt die Klasse `gameCard` noch eine Methode, die die Anzahl der "Treffer" zurückliefert. Das ist einfach eine vereinfachte Variante der bereits vorhandenen Methode `calculateWorth()`, die die Berechung des Wertes auslässt und stattdessen nur die Treffer zählt.
- Jede Karte wird nur einmal angeschaut, und die Anzahl der Karte wird zum Rätselergebnis addiert.
- Anschließend müssen die Zahlen der Nachfolgenden Zahlen gemäß der Regeln erhöht werden. Dazu bekommt die Klasse `gameCard` die Methode `public void erhoeheAnzahl(int gewonneneExemplare)`.
- Jetzt muss man nur noch für die nächsten "Anzahl-Treffer" Karten deren Anzahl um die Anzahl der eigenen Karte erhöhen. dabei muss man auch aufpassen, dass man am Ende der Kartenliste nicht aus der Liste plumpst - hier bietet sich die Kombination aus einem Iterator und einem Zähler an:

```
[...]

// Das erzeugt einen Iterator, der am Index "Kartennummer-1" losläuft.
Iterator<gameCard> it = cards.listIterator(c.getCardNo()-1);
// FIXME da fehlt noch was.

[...]

int steps=0;
while(it.hasNext() && steps < gewonneneKarten) {
   gameCard n = it.next();
   // FIXME hier fehlt was
   steps++;
}
```

Infos zu Iteratoren auf dieser Wiki-Seite

Lösungsvorschlag nach Variante 2

- day4.java
- gameCard.java

From:
https://info-bw.de/ -

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day4:start?rev=1701704928

Last update: **04.12.2023 15:48**