Tag 2

Aufgabe

- Aufgabe
- Inputdateien

:

- Beispiel d2e.txt
- Input d2i.txt.

Kontrollergebnisse

- Eingabedatei d2e.txt:
  - Teil 1 - 8
  - Teil 2 -
- Eingabedatei d6i.txt:
  - Teil 1 - 2286
  - Teil 2 - 74229

- Variante 1
- Variante 2: Objekte

Hilfestellungen Teil 1

- Du kannst/musst exzessiv die split("...")-Methode von Java nutzen. Damit zerteilst du einen String an jedem gefundenen Trennzeichen, sodass ein String-Array entsteht. Beispiel: `line.split(":")` erzeugt ein Array der Form ["Game 1", "4 red, 1 green, 15 blue; 6 green UND SO WEITER"]. Das Array enthält also zwei Strings: Erstens den Teil VOR dem Doppelpunkt, und zweitens den gesamten restlichen Teil hinter dem Doppelpunkt.
- Die split("...")-Methode kannst du mehrmals direkt hintereinander aufrufen und zwischendurch mit dem üblichen Index-Zugriff in eckigen Klammern auf einen bestimmten String des String-Arrays zugreifen. Beispiel: Der Befehl `line.split(":")[1].split(";")` gibt dir ein String Array, bei dem jeder String ein Herausziehen des Elfen darstellt. Das Ergebnis ist dann also z. B. die Form ["4 red, 1 green, 15 blue", "6 green, 2 red, 10 blue", "..."]
- Unterteile jeden dieser Strings noch einmal mehr (an einem weiteren Satzzeichen), um jede einzelne Farbe zu erhalten.
- Nun kannst du für eine einzelne Farbe (z. B. `3 green`) prüfen, ob diese Anzahl für diese Farbe erlaubt ist. Dazu kannst du den String ein weiteres Mal am Leerzeichen zerteilen und für den String-Vergleich `equals()` benutzen.
- Es empfiehlt sich immer wieder vorsorglich die `strip()` Methode zu nutzen. Diese entfernt Whitespace-Character (also z. B. ein Leerzeichen) am Anfang eines Strings und am Ende.
- Nutze eine boolean-Variable, um festzuhalten, ob du in der aktuellen Zeile bereits eine illegale Anzahl einer Farbe gezogen hast. Dann kannst du nämlich früher die Schleifen unterbrechen (`break;`) und weißt am Ende auch, ob denn die aktuelle Zeilennummer aufaddiert werden muss oder nicht.

Lösungsvorschlag Teil 1

```java
private boolean colorAmountAllowed(String s) {
    String text = s.strip().split(" ")[1].strip();
    int amount = Integer.parseInt(s.strip().split(" ")[0].strip());

    if (text.equals("red") && amount > 12) {
        return false;
    } else if (text.equals("green") && amount > 13) {
        return false;
    } else if (text.equals("blue") && amount > 14) {
        return false;
    }
    return true;
}

public int partOne() {
    int summe = 0;

    for (String line : inputLines) {
        // Speichere die Game ID
        int gameID = Integer.parseInt(line.split(":")[0].split(" ")[1]);
        boolean possibleGame = true;

        // Zerteile die line in die einzelnen "reveals"
        String[] reveals = line.split(":")[1].split(";");
        for (String reveal : reveals) {

            // Zerteile jeden Reveal in die einzelnen Farben
            String[] colors = reveal.split(",");
            for (String color : colors) {

                // Prüfe jede Farben-Häufigkeit-Kombination
                if (!colorAmountAllowed(color)) {
                    possibleGame = false;
                    break;
                }
            }
            if (!possibleGame) {
                break;
            }
        }

        if (possibleGame) {
            summe += gameID;
        }

    }

    return summe;
```

```
}
```

Hilfestellungen Teil 2

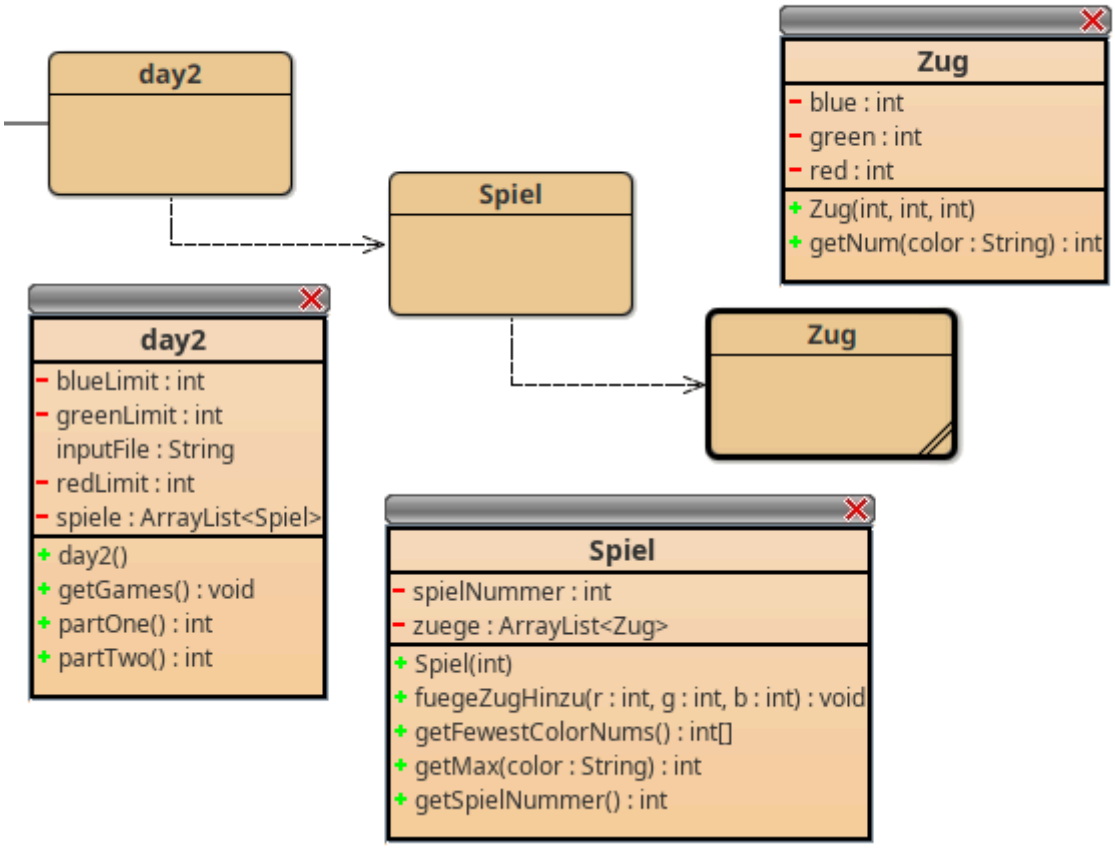
- Das meiste der verschachtelten Schleifen kannst du stehen lassen. Du benötigst nun nicht mehr die boolean-Variable, um zu überprüfen, ob die aktuelle Zeile 'erlaubt' ist. Jede Zeile fließt gleichermaßen in das Ergebnis ein.
- Erstelle pro Zeile drei int-Variablen, die dir das größte Vorkommen von rot, grün und blau Speichern. Für jede Farbe überprüfst du, ob die Zahl der jeweiligen Farbe größer ist als der bisher gespeicherte Wert in den neuen Variablen.
- Am Ende jeder Zeile fügst du das Produkt der drei Variablen zur Gesamtsumme hinzu.

Lösungsvorschlag Teil 2

```
int summe = 0;

for (String line : inputLines) {
    // Speichere die Game ID
    int gameID = Integer.parseInt(line.split(":")[0].split(" ")[1]);

    int greenMaxAmount = 1;
    int redMaxAmount = 1;
    int blueMaxAmount = 1;

    // Zerteile die line in die einzelnen "reveals"
    String[] reveals = line.split(":")[1].split(";");
    for (String reveal : reveals) {

        // Zerteile jeden Reveal in die einzelnen Farben
        String[] colors = reveal.split(",");
        for (String color : colors) {
            String text = color.strip().split(" ")[1].strip();
            int amount = Integer.parseInt(color.strip().split("
")[0].strip());

            if (text.equals("green") && amount > greenMaxAmount) {
                greenMaxAmount = amount;
            } else if (text.equals("red") && amount > redMaxAmount) {
                redMaxAmount = amount;
            } else if (text.equals("blue") && amount > blueMaxAmount) {
                blueMaxAmount = amount;
            }
        }
    }

    summe += greenMaxAmount * redMaxAmount * blueMaxAmount;

}
```

```
return summe;
```

Das Prinzip von Variante 2 ist dasselbe wie bei Variante 1, die zentrale Methode zur Aufteilung der Zeilen ist und bleibt `String.split()`. Variante 2 modelliert das Problem objektorientiert:



Anmerkung: Das ist natürlich Overkill - vor allem für die Züge eine eigene Klasse zu implementieren, die letztlich nur die Werte für RGB enthält - aber es zeigt das Prinzip der Objektorientierung dennoch ganz gut:

- Jede Klasse definiert einen neuen Datentyp, hier erhalten wir `Spiel` um Spiele mit ihren Zügen zu speichern, und Zug um einen Zug zu speichern.
- Die Klasse day2 verwaltet eine `ArrayList` des Typs `Spiel`: `private ArrayList<Spiel> spiele;`
- Die Klasse `Spiel` speichert alle Züge, die zu einem Spiel gehören in einer `ArrayList` des Typs Zug: `private ArrayList<Zug> zuege;`

Nun verlagert sich die Arbeit vor allem auf das Einlesen der Rätseldaten in der Methode `getGames()` - wenn man das hat, kann man so ziemlich alle Infos sehr einfach extrahieren.

`getGames()` macht prinzipiell folgendes:

- Splitte die Eingabezeile am Doppelpunkt, der erste Teil beinhaltet die Infos zur Spielnummer, wenn man die Spielnummer extrahiert hat, kann man das neue Spiel erzeugen.
- Der zweite Teil ist ein String mit allen Zügen zu diesem Spiel.

Tipp: Erster Split & Erzeugung des Spiel-Objekts

```
// Spielnummer
            String gameString = line.split(":")[0];
            int spielNummer = Integer.parseInt(gameString.split("Game
")[1]);
            // Neues Spiel instanziieren
            Spiel s = new Spiel(spielNummer);

            // String mit allen Zügen
            String drawString = line.split(":")[1];
```

- Die einzelnen Züge erhält man durch den Split des Strings mit allen Zügen am Semikolon, die einzelnen Farben dann, indem man die Substrings an den Kommata splittet:

Tipp: Gerüst mit Schleifen zu den Zugdetails

```
// Einzelne Züge holen.
String[] draws = drawString.split(";");
for(String d: draws) {
                // Die einzelnen Farben abarbeiten
                int nRed = 0;
                int nBlue = 0;
                int nGreen = 0;
                String[] colors = d.split(",");
                for(String c: colors) {
                    if(c.indexOf("green") != -1) nGreen =
Integer.parseInt(c.split("green")[0].trim());
                    if(c.indexOf("red") != -1) nRed =
Integer.parseInt(c.split("red")[0].trim());
                    if(c.indexOf("blue") != -1) nBlue =
Integer.parseInt(c.split("blue")[0].trim());
                }
                s.fuegeZugHinzu(nRed, nGreen, nBlue);

            }
```

- Wenn das Spiel dann komplett, muss es noch in die ArrayList eingefügt werden: `spiele.add(s);`

Die Lösungen für Teil 1 und 2 sehen in dieser Variante so aus:

Jetzt kann man direkt über die Spiele iterieren und alle Infos nach belieben auswerten.

```
public int partOne() {
        int answer=0;
        for(Spiel s: spiele) {
            if( s.getMax("red") <= redLimit &&
                s.getMax("green") <= greenLimit &&
```

```
                s.getMax("blue") <= blueLimit ) {
                    answer += s.getSpielNummer();
                }
        }
        return answer;
    }

    public int partTwo() {
        int answer=0;
        for(Spiel s: spiele) {
            int[] fewest = s.getFewestColorNums();
            answer += fewest[0] * fewest[1] * fewest[2];
        }
        return answer;
    }
```

Code zu dieser Variante: https://codeberg.org/qg-info-unterricht/aoc-bluej-2023 in den Dateien day2.java, Spiel.java und Zug.java.[1)]

1)
Dieser Commit enthält die Änderungen zu Day 1: https://codeberg.org/qg-info-unterricht/aoc-bluej-2023/commit/5687838212e0e5bde7ff836fbc7847343b047610

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

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day2:start

Last update: **12.11.2024 06:32**