Tag 2

- Variante 1
- Variante 2: Objekte

Hilfestellungen Variante 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

```
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

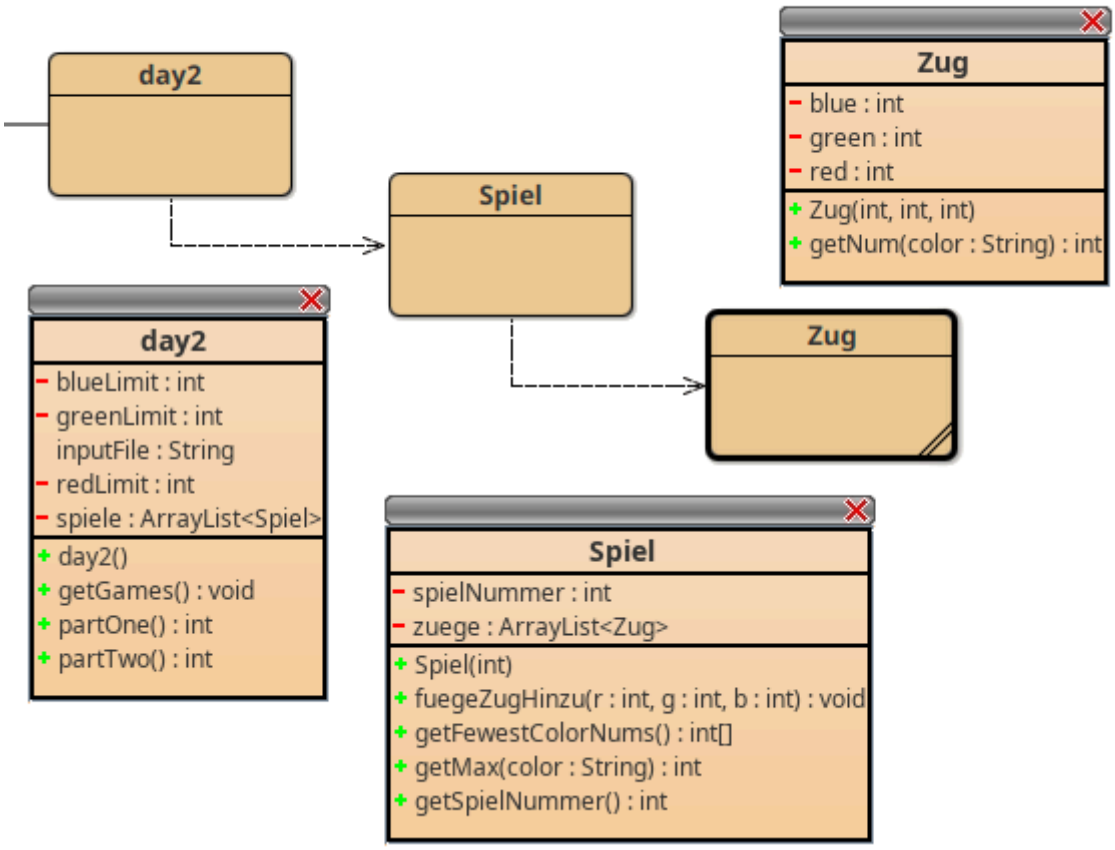
- Das meiste der verschachtelten Schleifen kannst du stehen lassen. Du benötigst nun nicht mehr die boolean-Variable, um zu überprüfen, ob die aktuelle Zeile 'erlaubt' ist. Jede Zeile fliest gleichermaßen in das Ergebnis ein.
- Erstelle pro Zeile drei int-Variablen, die dir das größte Vorkommen von rot, 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 eigen Klasse zu implementieren, die letztlich nur die Werte für RGB enthält - aber es zeigt das Prinzip dennoch ganz gut.

Nun verlagert sich die Arbeit vor allem auf das Einlesen der Rätseldaten - wenn man das hat, kann man so ziemlich alle Infos sehr einfach in kurzer Zeit wieder rausholen:

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

<code java> 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