Tag 3

- Variante 1

Hilfestellungen Teil 1

- Fasse die Eingabe als zweidimensionales Arrays auf! Jeder Character hat eine Koordinate, die man mit x und y beschreiben kann. (y beginnt in der Informatik typischerweise links oben → das passt: y=0 ist die erste/oberste Zeile).
- Nutze daher auch NICHT die foreach-Schleife, sondern eine übliche for-Schleife, in der du für jede Zeile y erhöhst. Speichere die aktuelle Zeile am besten in einer Variablen (z. B. `line`).
- Prüfe Reihe für Reihe von links nach rechts für jeden Character, ob dieser eine Ziffer ist (`Character.isDigit(line.charAt(x))` **und** ob er *Symbole als Nachbarn* hat.
- Die Prüfung der *Symbole als Nachbarn* lagern wir in eine separate Methode aus, die die x- und y-Koordinate der zu überprüfenden Ziffer entgegennimmt. Prüfe für jede benachbarte Koordinate (dx von x-1 bis x+1 sowie dy von y-1 bis y+1), ob der dortige Character c weder eine Ziffer (siehe vorheriger Punkt) noch ein Punkt (`c != '.'`) ist. Ist es weder das eine, noch das andere, dann kann insgesamt true zurückgegeben werden, da mindestens ein Nachbar ein Symbol ist.
- Nun weiß man, dass die Ziffer der aktuellen x- und y-Koordinate Teil einer größeren Zahl ist, die vollständig erfasst werden muss. Auch diese Erfassung der gesamten Zahl anhand einer Ziffern-Koordinate lagern wir wieder in eine separate Methode aus (z. B. getFullNumber(x, y).
- **getFullNumber(x, y):** Zunächst müssen wir zum Ende der gesamten Zahl gehen - wir werden dann von hinten nach vorne alle Ziffern der Zahl nacheinander erfassen. Da wir auch zu einem späteren Zeitpunkt erneut an das Ende der Zahl müssen, bietet es sich an, auch diesen Punkt in eine separate Methode auszulagern:
- **getEndOfNumber(x, y):** Solange die x-Koordinate kleiner als die Zeilenlänge ist **und** der nachfolgende (rechte) Character eine Ziffer ist, solange wird x um eins erhöht. Dann wird x zurückgegeben und gibt die hinterste/rechteste Ziffer der gesuchten Zahl an.
- Jetzt wieder zurück in **getFullNumber(x, y):** Wir benötigen eine Variable, die zunächst auf die hinterste Ziffer zeigt (`end`). Solang end > 0 **und** solange der aktuelle Character eine Ziffer ist, solange wird der aktuelle Character c als Zahl aufgefasst (`Character.getNumericValue©`) und mit passender Wertigkeit (um Faktor 10 erhöhen pro Stelle) zum Zahlenwert dazuaddiert. Man kann der Reihe nach also z. B. 5*10^0 + 3*10^1 + 7*10^2 rechnen. Die Potenz kann berechnet werden mit `Math.pow(10, stelle)`, wobei `stelle` anfangs 0 ist und jeweils um 1 erhöht wird.
- Die gesamte Zahl ist damit fertig erfasst und wird in die Hauptmethode zurückgegeben und dort zur Gesamtsumme aufaddiert.
- Nun ist es abschließend noch wichtig den aktuellen x Wert an das Ende der aktuellen Zahl zu setzen. Denn andernfalls würde man x nur um 1 erhöhen und für den nächsten Character überprüfen, ob dieser eine Ziffer ist und auch Nachbarn hat. Dabei kann x sich aber noch immer in derselben Zahl befinden, die soeben bereits erfasst wurde. Wir müssen also sicherstellen, dass wir diese Zahl nicht erneut überprüfen. Daher rufen wir nochmal `getEndOfNumber(x, y)` auf und setzen x auf das Ergebnis dieser Methode **+ 1**.

Lösungsvorschlag

```
/**
 * Prüft, ob die Zahl an der angegebenen Koordinate einen validen Nachbarn
```

```
hat (ein Symbol).
 */
private boolean numberHasNeighbors(int x, int y) {
    for (int dx = x - 1; dx <= x+1; dx++) {
        for (int dy = y - 1; dy <= y+1; dy++) {
            if (dx < 0 || dy < 0 || dx >= inputLines.get(0).length() || dy
>= inputLines.size()) {
                continue;
            }
            char c = inputLines.get(dy).charAt(dx);
            if (c != '.' && !Character.isDigit(c)) {
                return true;
            }
        }
    }
    return false;
}

/**
 * Gibt die letzte x-Koordinate der Zahl zurück.
 */
private int getEndOfNumber(int x, int y) {
    String line = inputLines.get(y);
    while(x < line.length() - 1 && Character.isDigit(line.charAt(x+1))) {
        x++;
    }
    return x;
}

/**
 * Eingabe: eine Koordinate
 * Ausgabe: die gesamte Zahl, die sich möglicherweise auch
 *          links und rechts von dieser einen Koordinate versteckt.
 */
private int getFullNumber(int x, int y) {
    String line = inputLines.get(y);

    // geh zum rechten Ende der Zahl
    int end = getEndOfNumber(x, y);

    // gehe von rechts zurück bis zum Anfang der Zahl und setze sie dabei
zusammen
    int zahl = 0;
    int stelle = 0;
    while (end >= 0 && Character.isDigit(line.charAt(end))) {
        zahl += Character.getNumericValue(line.charAt(end)) * Math.pow(10,
stelle);
        stelle++;
        end--;
```

```
    }

    return zahl;
}

public int partOne() {
    int summe = 0;

    // y for line
    for (int y = 0; y < inputLines.size(); y++) {
        String line = inputLines.get(y);
        // i for index
        for (int x = 0; x < line.length(); x++) {
            if (Character.isDigit(line.charAt(x)) && numberHasNeighbors(x,
y)) {
                int number = getFullNumber(x, y);
                summe += number;
                x = getEndOfNumber(x, y) + 1;
            }
        }

    }

    return summe;
}
```

Hilfestellungen Teil 2

- Du kannst zwei Methoden aus Teil 1 wiederverwenden! Gehe Zeile für Zeile über jeden Character. Wenn der Character c ein Stern ist (`c == '*'`), dann lässt du dir **alle** benachbarten Zahlen geben (das kann nur eine, oder zwei, oder drei, … Zahlen sein).
- Die Erfassung aller benachbarten Zahlen kannst du in eine separate Methode auslagern. Dort nutzt du eine ArrayList, um die variable Anzahl aller benachbarten Zahlen abzuspeichern (`ArrayList numbers = new ArrayList<Integer>();`). Gehe nun Reihe für Reihe (!!!) alle benachbarten Character mit dx und dy ab. Die äußere Schleife muss also über dy gehen! Wenn ein benachbarter Character eine Ziffer ist, dann kannst du direkt die Methode `getFullNumber(dx, dy)` aus Teil 1 nutzen und die Zahl zu `numbers` hinzufügen. **Wichtig:** du musst nun dx aufs Ende der Zahl setzen, damit die Zahl nicht ein zweites Mal als Nachbar erkannt wird (`dx = getEndOfNumber(dx, dy);`).
- Wenn die gefüllte ArrayList wieder in der Hauptmethode ankommt, dann musst du überprüfen, wie viele benachbarte Zahlen es enthält. Wenn es nicht genau 2 sind, dann geschieht nichts. Andernfalls wird das Produkt beider Zahlen zum Endergebnis hinzugefügt.

Lösungsvorschlag

```
private ArrayList<Integer> getAdjacentNumbers(int x, int y) {
    ArrayList numbers = new ArrayList<Integer>();
    for (int dy = y-1; dy <= y+1; dy++) {
        for (int dx = x-1; dx <= x+1; dx++) {
            if (dx < 0 || dy < 0 || dx >= inputLines.get(0).length() || dy
```

```
>= inputLines.size()) {
                    continue;
                }

                String line = inputLines.get(dy);
                if (Character.isDigit(line.charAt(dx))) {
                    numbers.add(getFullNumber(dx, dy));
                    dx = getEndOfNumber(dx, dy);
                }

            }
        }
        return numbers;
    }

    public int partTwo() {
        int summe = 0;

        for (int y = 0; y < inputLines.size(); y++) {
            String line = inputLines.get(y);
            for (int x = 0; x < line.length(); x++) {
                if (line.charAt(x) == '*') {
                    ArrayList<Integer> numbers = getAdjacentNumbers(x, y);
                    if (numbers.size() == 2) {
                        summe += numbers.get(0) * numbers.get(1);
                    }
                }
            }
        }
        return summe;
    }
```