Tag 6: Wait For It

Aufgabe

- Aufgabe

Kontrollergebnisse

- Eingabedatei d4e.txt: [Teil 1 -] [Teil 2 -]
- Eingabedatei d4i.txt: [Teil 1 -] [Teil 2 -]

- Variante 1
- Variante 2: Etwas Mathematik?

Hilfestellung Teil 1

- Es verschiedene Möglichkeiten die wenigen Zahlen zu speichern. In dieser Lösung empfehle ich, alle Zahlen der ersten Zeile und alle Zahlen der zweiten Zeile jeweils in einem int-Array zu speichern. Diese Arrays kannst du z. B. `times` und `distances` nennen. Danach kannst du mit einer for-Schleife über die Werte iterieren. Mit deinem Laufindex (z. B. `i`) kannst du sicherstellen, dass du immer auf die zueinander gehörenden Zeiten und Distanzen zugreifst.
- Für jedes Wertepaar (Zeit und Distanz) musst du nun für jede mögliche 'Knopf-Drück-Zeit' von 0 bis 'Zeit' ms prüfen, ob die zurücklegbare Strecke größer als die eingelesene Distanz ist. Für die zurücklegbare Strecke gilt nach den Regeln der Physik: `s=v*t`. `v` ist die Geschwindigkeit, welche sich direkt aus der 'Knopf-Drück-Zeit' ergibt. `t` ist dann die noch übrige Zeit, um die Strecke zurückzulegen (also die eingelesene Zeit *minus* 'Knopf-Drück-Zeit'). Wenn das Ergebnis `v` größer ist als die eingelesene Distanz, dann wird der Counter für das Wertepaar um eins erhöht.
- Am Ende müssen alle Counter zusammenmultipliziert werden.

Lösungsvorschlag

```
private int[] numbersToIntArray(String numbersStr) {
    String[] numbers = numbersStr.trim().split("\\s+");
    int[] result = new int[numbers.length];

    for (int i = 0; i < numbers.length; i++) {
        result[i] = Integer.parseInt(numbers[i]);
    }

    return result;
}



public int partOne() {
    int[] times = numbersToIntArray(inputLines.get(0).split(":")[1].trim());
```

```
    int[] distances =
numbersToIntArray(inputLines.get(1).split(":")[1].trim());

    int result = 1;

    for (int i = 0; i < times.length; i++) {
        int numberOfWaysToBeat = 0;
        for (int buttonTime = 0; buttonTime < times[i]; buttonTime++) {
            if ((buttonTime * (times[i]-buttonTime)) > distances[i]) {
                numberOfWaysToBeat++;
            }
        }
        result *= numberOfWaysToBeat;
    }


    return result;
}
```

Hilfestellung Teil 2

- Teil 2 ist tatsächlich noch einiges einfacher als Teil 1. Dadurch, dass die erste und zweite Zahl jeweils als eine Zahl interpretiert wird, kann das Array wegfallen und damit kann auch eine Schleife wegfallen.
- Wichtig ist nun allerdings, dass man beide Zahlen als `long` speichert, da die Zahlen den Wertebereich von int andernfalls überschreiten können.
- Um aus den ersten Zahlen einer Zeile eine gemeinsame Zahl zu machen bevor man sie parsed, kann man alle Leerzeichen zwischen den Ziffern entfernen mit `replaceAll(" ","")`. Damit wird jedes Leerzeichen durch 'nix' ersetzt.

Lösungsvorschlag

```
public long partTwo() {
    long time = Long.parseLong(inputLines.get(0).split(":")[1].replaceAll(" ", ""));
    long distance =
Long.parseLong(inputLines.get(1).split(":")[1].replaceAll(" ", ""));

    long numberOfWaysToBeat = 0;
    for (long buttonTime = 0; buttonTime < time; buttonTime++) {
        if ((buttonTime * (time-buttonTime)) > distance) {
            numberOfWaysToBeat++;
        }
    }

    return numberOfWaysToBeat;
}
```

Variante 2: Mitternachtsformel

Wenn man unter der Dusche etwas nachdenkt, stellt mal fest, dass das eigentlich eine Matheaufgabe ist: Man kann berechnen, wie weit man kommt, je nach dem wie lang man den Knopf drückt und wenn man weiß, welche Zeit insgesamt zur Verfügung steht.

Fragen:

- Wie weit kommt man in der Zeit 17ms, wenn man den Knopf 3ms drückt?
- Wie sieht es bei 37ms und 13ms gedrücktem Knopf aus?
- Wie bei `t` ms und p ms gedrückten Knopf?

Tipp 1:

Wenn man p ms lang den Knopf drückt, kann sich das Boot `t-p` ms lang mit der Geschwindigkeit p bewegen, kommt also

```
d_t(p) = (t-p)*p
```

Millimeter weit. Für jedes gegebene `t` und p ergibt sich eine andere Entfernung.

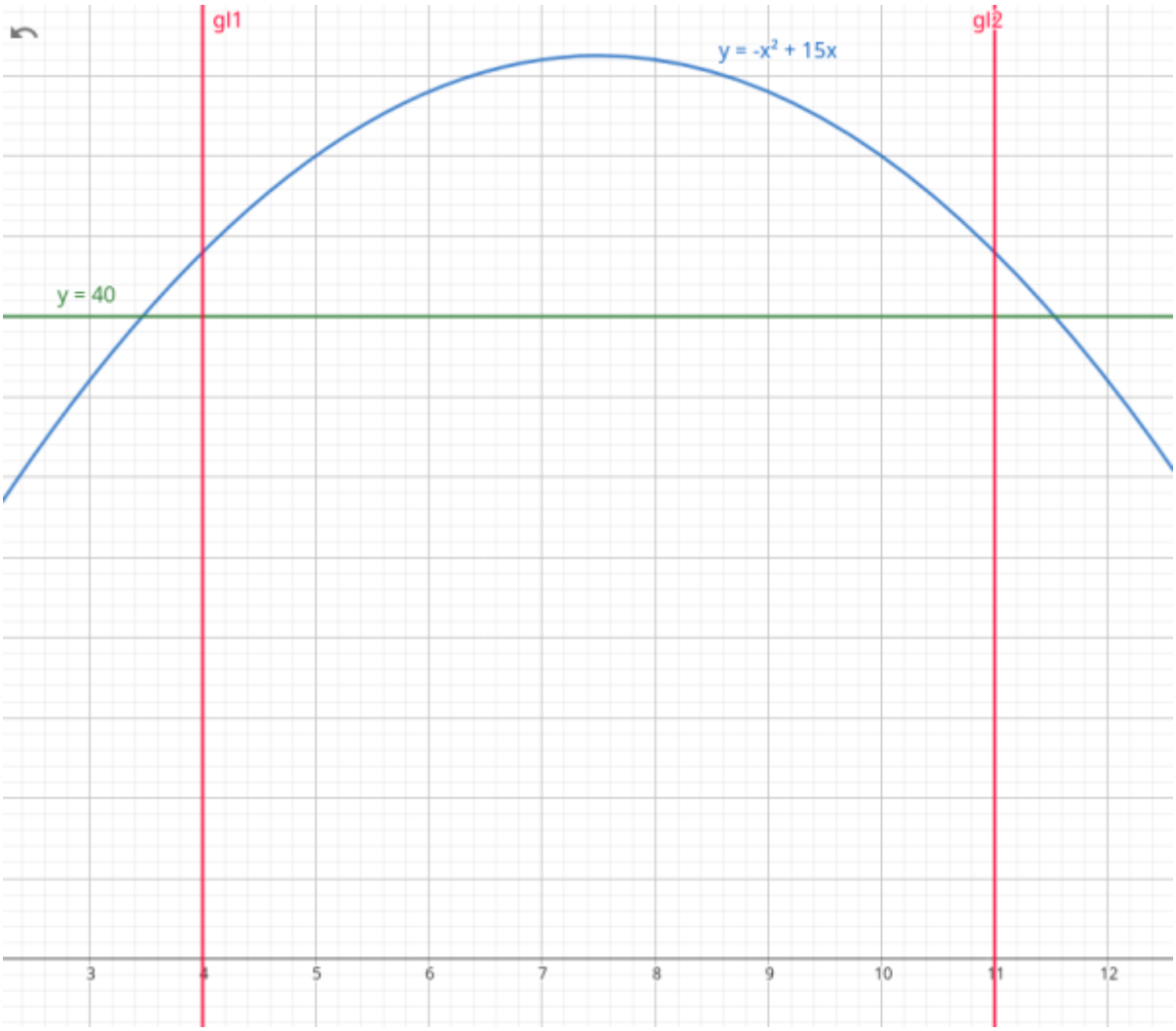
Multipliziere deine Formel aus, was erhältst du dann?

Tipp 1:

```
d_t(p) = (t-p)*p = -p² + tp
```

Eine nach unten geöffnete Parabel.

Für das zweite Wertepaar `t=15` und `d=40` kann man mit diesen Überlegungen folgendes Bildchen zeichnen, in dem man ablesen, kann dass man mindestens 4 und höchstens 11 ms drücken muss, um den bisherigen Rekord von 40mm zu überbieten



Nun kann man die Aufgabe mit der Mitternachtsformel und den beiden Methoden `Math.floor` und `Math.ceil` direkt lösen, ganz ohne Schleifen und das klappt dann auch für beide Teile, wenn man aus der Aufgabe des Vortags noch mit long Variablen hantiert, sonst muss man den Typ für Teil 2 noch umstellen.

Aufpassen muss man noch bei Fällen wie dem dritten Beispiel, wenn die Mitternachtsformel direkt die Ränder des Bereichs liefert - die gehören nämlich nicht dazu, weil das Boot echt weiter als das bisherige Maximum fahren soll, gleich weit gehört nicht dazu. Hier muss man den Fall, dass die MNF "aufgeht" noch extra abfangen.

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

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day6:start?rev=1701876957

Last update: **06.12.2023 15:35**