Einführung

Um den Einstieg zu erleichtern, gibt es ein BlueJ-Template, das bereits Funktionalitäten mitbringt, um die Textdateien des AOC einzulesen.

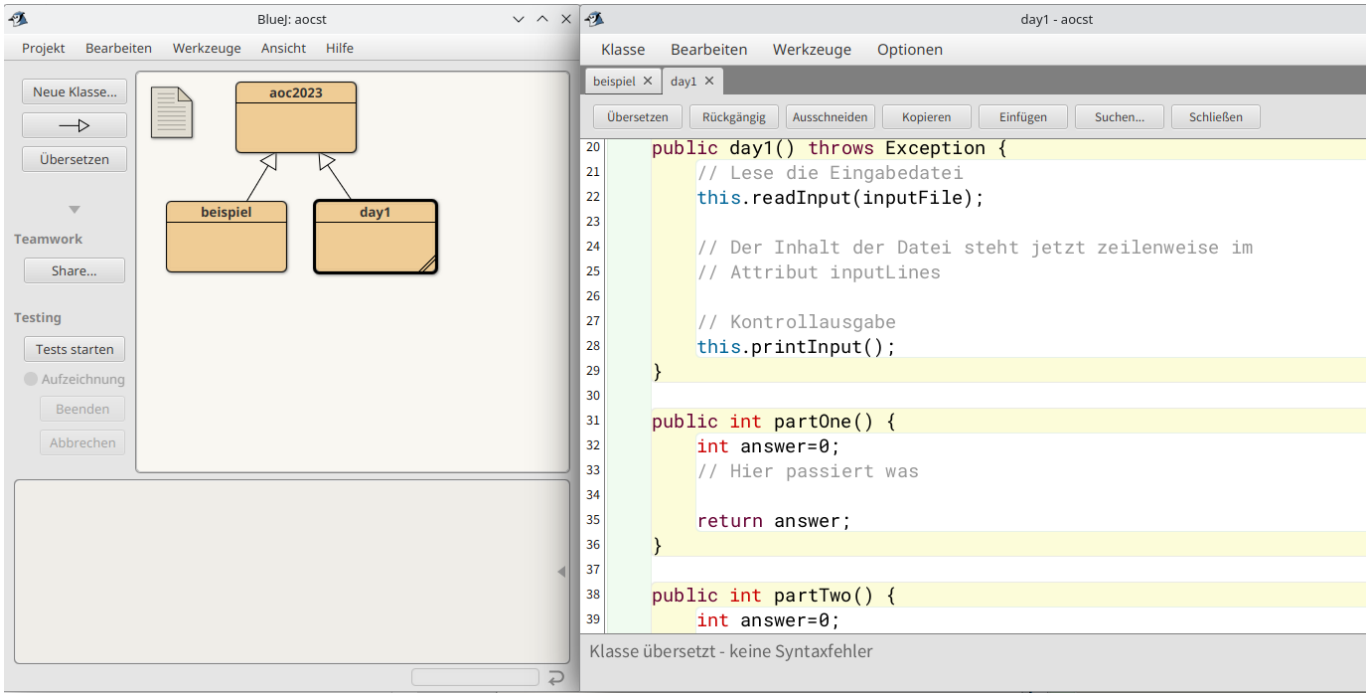
https://codeberg.org/qg-info-unterricht/aoc-starter-template

Vorschlag: Verwendung der Vorlage

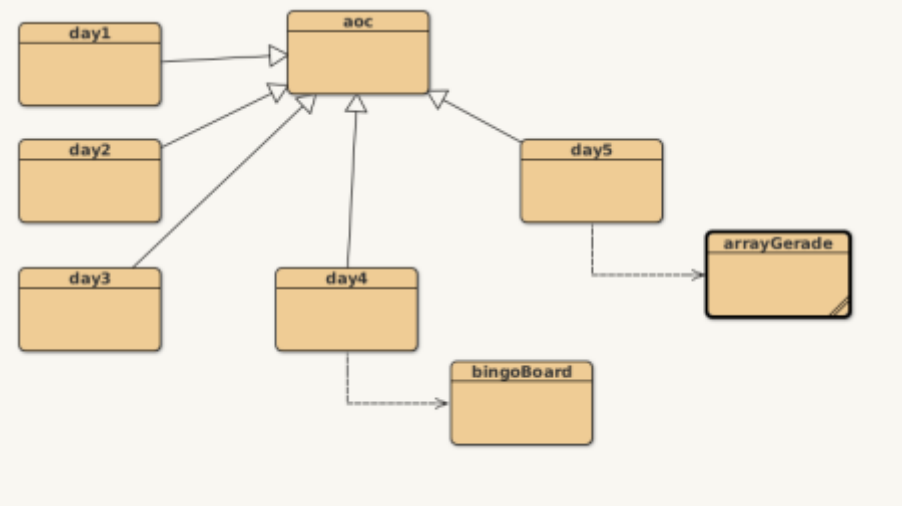
Eine Möglichkeit, diese Vorlage zu verwenden, ist es, für jeden Tag eine Subklasse zu erstellen. Auf diese Weise erbt man die Basis-Methoden `readInput(String filename)` und `printInput()` von der Superklasse, wenn weitere Methoden hinzukommen, die alle weiteren Tagsklassen gemeinsam haben sollten, kann man diese in der `aoc2023`-Klasse hinzufügen.

Außerdem kann man in der "Tagesklasse" jeweils die Methoden `partOne()` und `partTwo` erstellen - plus weitere Hilfsmethoden - um die Rätsel zu lösen.[1)]

Die Situation in BlueJ sieht dann so aus:



Wenn der AOC voranschreitet, kann das dann evtl. auch irgendwann so (oder so ähnlich) aussehen:



Tipp: Verwendung der AOC-Beispiele

Im "Aufgabentext" der AOC Aufgaben wird die Problemstellung ausführlich anhand eines *Beispiels* erläutert. Es ist meist eine gute Idee, die Lösung anhand dieses meist überschaubaren Beispiels zu implementieren, weil man hier sein eigenes Vorgehen besser nachvollziehen kann. Dazu muss man die Beispieleingabe als Datei ablegen und kann dann mit programmieren loslegen.

Wichtig: Man sollte unbedingt einen Blick in den "echten" Puzzle-Input werfen, um nicht mit falschen Voraussetzungen zu denken und zu programmieren, denn oft ist dieser zwar ähnlich hat aber mehr Zeilen, mehr Stellen, größere Koordinatenwerte u.ä. als das Beispiel.

Vorschlag: Die Daten des Beispiels in eine Datei mit dem Namen `d<Tagesnummer>e`, die des Inputs in `d<Tagesnummer>i` und dann im Attribut `inputFile` anpassen. Entwickeln mit `d<Tagesnummer>e`, Puzzle lösen mit `d<Tagesnummer>i`. In der Vorlage sind zwei beispielhafte Dateien hinterlegt.

Das Eingabeformat der Vorlage

Die Methdode `readInput(String filename)` liest die Daten aus der Eingabedatei zeilenweise in eine ArrayList von String-Arrays ein. Leere Zeilen werden dabei als leerer String eingelesen.

Die Sache mit den Zahlen

Wie in obigen Beispiel zu sehen, ist die Eingabe zunächst ein Array von Strings, was ungünstig ist, wenn man mit den Werten rechnen möchte/muss. Um aus den Strings Integer-Werte zu machen, verwendet man `Integer.parseInt(String)`:

```
for (String[] line: input) {
   // line ist ein Array der Länge 2
   //String ersteZahl  = line[0]);
   int ersteZahl = Integer.parseInt(line[0])
   // String zweiteZahl = line[1]);
   int zweiteZahl = Integer.parseInt(line[1])
   // ... rechne wat ...
}
```

Das Beispiel aus AOC 2022 in der BlueJ-Vorlage

Die BlueJ-Vorlage kommt mit einem Beispiel aus AOC 2022:

- Aufgabe

In der `beispiel`-Klasse ist eine mögliche Lösung für Teil 1 implementiert, dort kann man auch die Umwandlung von Strings in Zahlen nachvollziehen. Das kann man mal ausprobieren, ändern und vielleicht versuchen die Lösung für Teil 2 noch nachzutragen. In der Datei `bsp1e` sind die Zahlen aus dem AOC-Erklärungs-Beispiel enthalten.

Die Ergebnisse für die Eingaben, die in der Datei `bsp1i` in der BlueJ-Vorlage enthalten ist sind die folgenden:

Ergebnis Teil 1 für die Eingabe auf dieser Wikiseite

70509

Ergebnis Teil 2 für die Eingabe auf dieser Wikiseite

208567

Mehrere Felder Trennen

Teilweise ist es nötig, die eingelesenen Zeilen an bestimmten Stellen nochmals zu trennen, wenn die Zeilen z.B so aussehen

```
0 -> 802
2 -> 900
4 -> 1002
```

möchte man häufig die Wertepaare (0,802), (2,900), (4,1002) erhalten.

Hier muss man die Zeilen dann für die weitere Problemlösung vorbereiten. Interessante Methoden der String-Klasse dafür sind:

- `split(zeichen)`

- `replace(needle,subst)`
- `trim()`

[1)]

Weitere Tage kann man auch einfach durch kopieren von `day1` erstellen.