Einführung

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

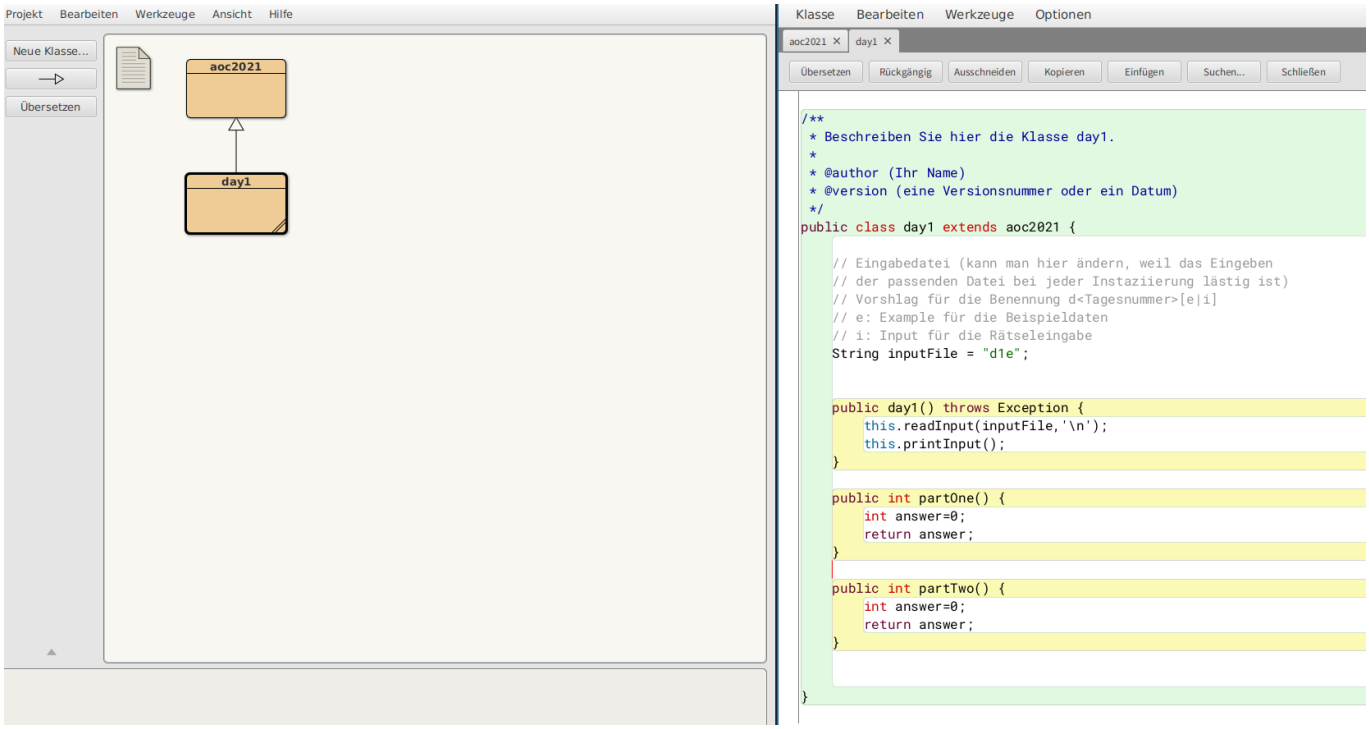
https://codeberg.org/qg-info-unterricht/aoc-starter-template

Vorschlag: Verwendung der Vorlage

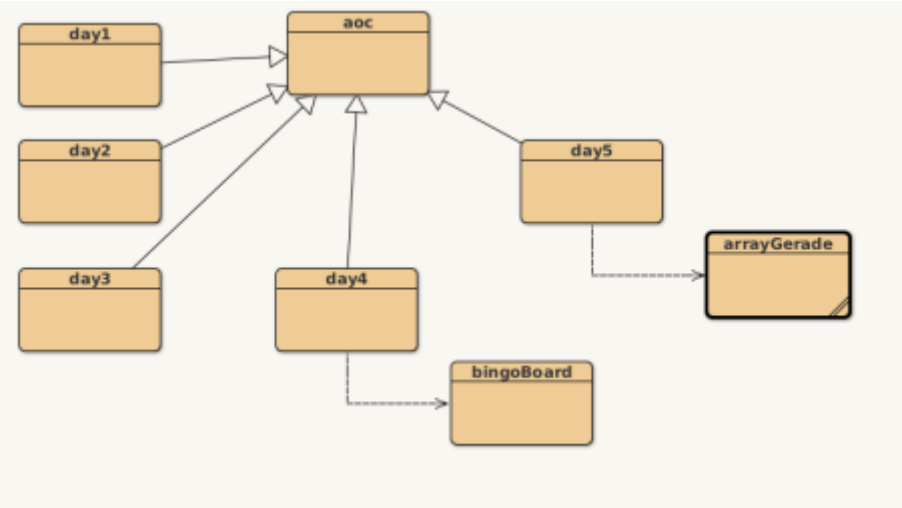
Eine Möglichkeit, diese Vorlage zu verwenden, ist es, für jeden Tag eine Subklasse zu erstellen. Auf diese Weise erbt man die Basis-Methoden `readInput(String filename)` und `printInput()` von der Superklasse, wenn weitere Methoden hinzukommen, die alle weiteren Tagsklassen gemeinsam haben sollten, kann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

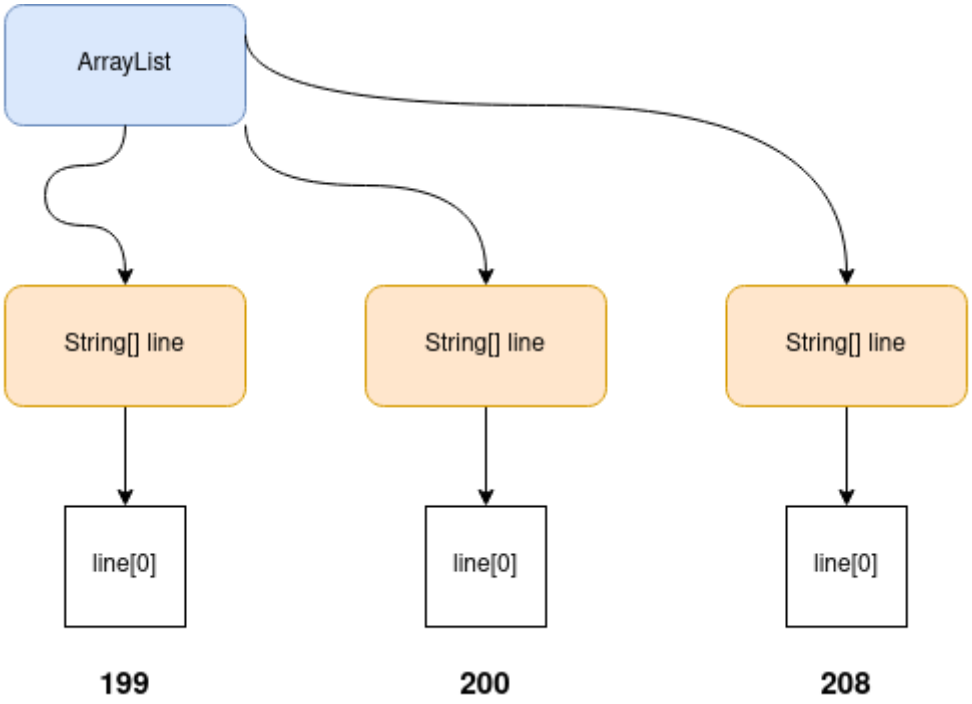
Im "Aufgabentext" der AOC Aufgaben wird die Problemstellung ausführlich anhand eines Beispiels erläutert. Es ist meist eine gute Idee, die Lösung anhand dieses meist überschaubaren Beispiels zu implementieren, weil man hier sein eigenes Vorgehen besser nachvollziehen kann.

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.

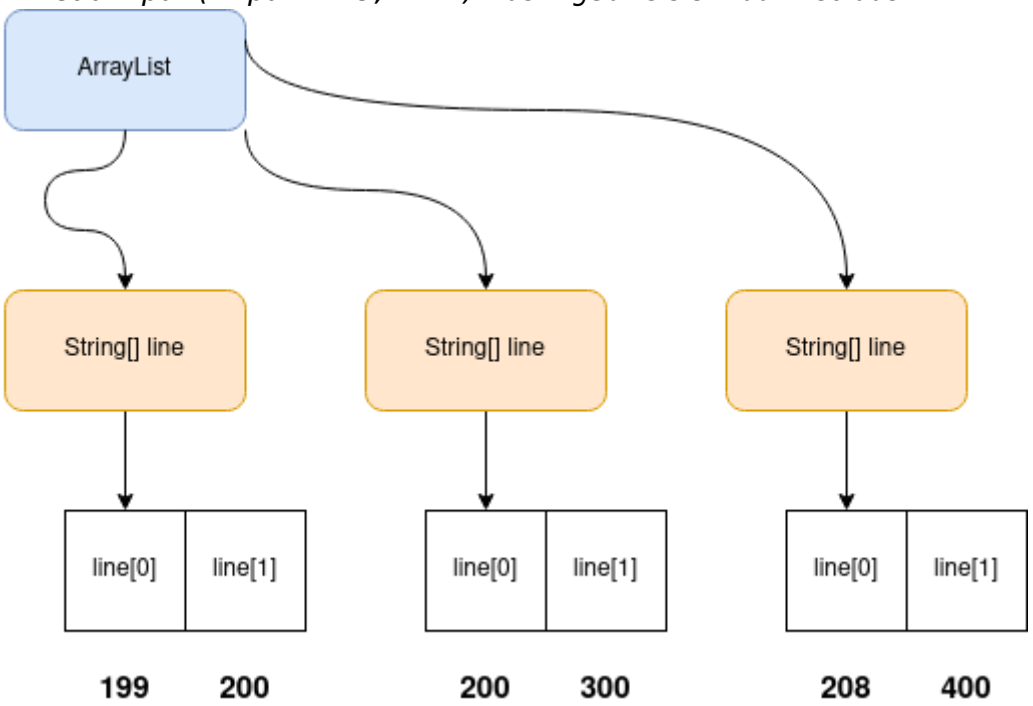
Vorschlag: Die Daten des Beispiels in eine Datei mit dem Namen `d<Tagesnummer>e`, die des Inputs in `d<Tagesnummer>i` und dann im Attribut `inputFile` anpassen. Entwickeln mit `d<Tagesnummer>e`, Puzzle lösen mit `d<Tagesnummer>i`. In der Vorlage sind zwei beispielhafte Dateien hinterlegt.

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. Man kann beim Aufruf der Methode das Trennzeichen angeben, an dem die Felder des Arrays getrennt werden. ==== Beispiel 1 - Jede Zeile hat einen Wert: ==== <code> 199 200 208 </code> Man trennt die Array-Felder am Newline-Zeichen` \n: readInput(inputFile, '\n') `Das Ergebnis sieht dann so aus:`



Die Zahlen ausgeben kann folgendermaßen: <code java> for (String[] line: input) { *line ist ein Array der Länge 1 mit nur einem Feld System.out.println(line[0]); } </code> ==== *Beispiel 2 - Zeilen mit mehreren Werten:* ==== <code> *199 200 200 300 208 400* </code> *Man trennt die Array-Felder an Leerzeichen \n:* `readInput(inputFile, ' ')` *Das Ergebnis sieht dann so aus:*



Zugriff folgendermaßen: <code java> for (String[] line: input) { line ist ein Array der Länge 2 String ersteZahl = line[0]); String zweiteZahl = line[1]); *... tu wat ... } </code> ==== *Trennzeichen* ==== *Trennzeichen kann nur ein Char-Wert sein, man kann also auch an* `-` *oder* `|` *trennen, aber nicht an* `->`[2] *- wenn man damit die Felder nicht passend aufbereiten kann, benötigt man eine weitere Methode, um die input-ArrayList weiter für die Problemlösung vorzubereiten. Interessante Methoden der String-Klasse dafür sind:* * `split(zeichen)` * `replace(needle,subst)` * `trim()` ===== *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)`*: <code java> 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 … } </code>

[1]

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

[2]

Zwei Zeichen…

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

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:java:aoc:einfuehrung:start?rev=1700823460

Last update: **24.11.2023 10:57**