

Sprachkonzepte

Erik Manz, Chris Jakob (WiSe 2023/24 AIN)

Aufgabe 1)

a)

Wir suchten als erstes die Dokumentationen heraus und haben währenddessen angefangen bezüglich util.Formatter die Vorgabe %(flags)(width)(conversion) zu verstehen. Später haben wir dann noch die Fälle mit (.precision) und (argumentIndex) hinzugefügt.

Für jede einzelne Kategorie haben wir uns einen Regex String im Java Code entwickelt und mithilfe von regex101.com auf Gültigkeit überprüft.

Anschließend haben wir unsere Regex Kategorien zusammengefügt und mit einem Pattern in einem Matcher auf den Input initialisiert.

Folgend laufen wir durch eine while Schleife so lange matcher.find() und prüfen dann welcher Regex dem matcher.group.matches() entspricht.

Die Ergebnisse werden in einen Stringbuilder geschrieben, den wir am Programmende ausgeben.

```
String fullRegex = „(%([1-9]\\$)*[-#\\+ 0,\\(\\)*[0-9]*\\.\\d*)?([AaBbCcDdEefGghHnoSsXx%]|([tT][HhIiKkLlMmSsLlNnpzKsQqBbhAaCcYyjmdeRrTtDdFfCf]?)))([a-zA-Z:]+)“;
```

Der gesamte Code liegt dem im Anhang beigefügten Archiv bei.

Beispielsausgabe

```
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPKO> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPKO\Aufgabe1\target\classes' 'org.example.App'
Aufgabe 1 Input:

Wochentag: %tA Uhrzeit: %tT
TEXT(Wochentag: )FORMAT(%tA)TEXT( Uhrzeit: )FORMAT(%tT)
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPKO> |
```

b)

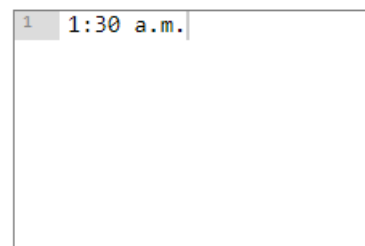
Wir nutzen <http://lab.antlr.org/> um unsere Regeln der Grammatik zu definieren und auszuführen.

Hier gibt es auf der linken Seite zwei Tabs mit Eingabefeldern: Einmal für die Grammatik und einmal für den Parser.

Dies sind unsere Regeln für den Parser:

```
//INTO PARSER
```

```
parser grammar ExprParser;
options { tokenVocab=ExprLexer; }
start: TIME;
```



Start rule ?

start Run Show profiler

Tree Hierarchy

```
start:1
|
1:30 a.m.
```

Und dies sind unsere Regeln für den Lexer:

```
//INTO LEXER TAB:
lexer grammar ExprLexer;

fragment HOUR: ('0'?[0-9] | '1'[0-2]);
fragment HOUR23: ( ('0'? | '1') [0-9] | '2'[0-3]);
fragment HOUR2400: ('24:00');
fragment SEP: ':';
fragment MIN: [0-5][0-9];
fragment WHITE: ' ';
fragment STR: ('a.m.' | 'p.m. ');
fragment TWLF: '12'(' midnight' | ' noon');
fragment ONLYSTR: ('Noon' | 'Midnight');
TIME: (HOUR SEP MIN WHITE STR) | (HOUR23 SEP MIN) | HOUR2400 | TWLF | ONLYSTR;
```

Diese Regeln haben wir via Trial and Error verfeinert. Hierzu ist das Input Feld sehr hilfreich, und das Start Rule Feld muss dafür, dass es funktioniert, richtig eingetragen sein. In unserem Fall: „start“

Aufgabe2

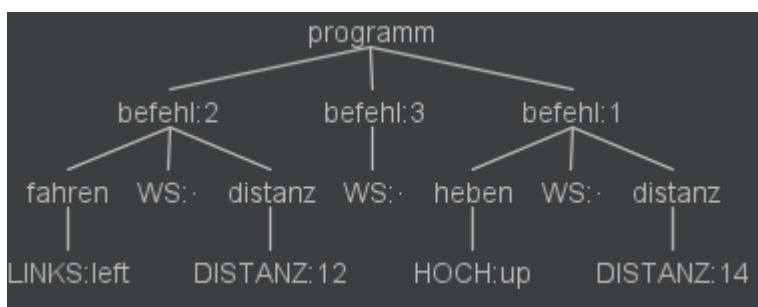
a)

Für die zweite Aufgabe haben wir uns eine Sprache zur Steuerung eines Gabelstaplers ausgedacht. Unsere Anforderungen an die Sprache waren, dass der Gabelstapler fahren, heben und lenken kann. Zuerst haben wir den Lexer mit den notwendigen Befehlen definiert. Der Lexer enthält die Befehle HOCH: 'up', RUNTER: 'down'; LINKS: 'left'; RECHTS: 'right'; VORWAERTS: 'forward'; RUECKWAERTS: 'backward'; HALT: 'stop'; DISTANZ: [0-9]+ und WS: ' '; mit denen der fiktive Gabelstapler gesteuert werden kann. Unsere erste Überlegung war, den Gabelstapler so lange fahren zu lassen, bis er anhält. Dies erwies sich jedoch als zu einfach, weshalb wir uns für die Methode mit Distanzangabe entschieden.

Für den Parser haben wir uns überlegt, dass der Gabelstapler immer eine bestimmte Strecke fahren, und dann etwas heben oder absetzen soll. Ein Befehl würde also so aussehen: forward 20 up 1000. Der Gabelstapler würde also 20 m geradeaus fahren und dann einen Gegenstand 1 m hochheben. Alternativ ist es auch möglich, dass der Stapler einen Gegenstand zuerst nach oben bewegt und dann fährt, z.B. down 1000 backward 10, dann würde der Stapler den Gegenstand zuerst 1 m nach unten bewegen und dann 10 m zurückfahren. Es können aber auch beliebig viele Befehle aneinandergereiht werden.

Der Parser und der Lexer befinden sich in der Zip-Datei, die unserem Bericht beigelegt ist.

Hier ist der Ableitungsbaum für left 12 up 14 in IntelliJ:



b)

Für den zweiten Teil der Aufgabe haben wir in Java die Klassen Befehl, Programm, Fahren und Heben erstellt und für jede einen Konstruktor und die notwendigen Getter und Setter angelegt. Dann haben wir die Klasse StaplerToAst erstellt, die die Main-Klasse enthält und den Stapler Builder aufruft, der die Grammatik anwendet und einen abstrakten Syntaxbaum erstellt.

In der Klasse StaplerBuilder haben wir den von antler erstellten StaplerParserBaseListener erweitert und dann die notwendigen Exit-Methoden überschrieben. Bei der Methode exitFahren sind wir so vorgegangen, dass wir ctx.start.getType() aufgerufen haben und dann in einem switch case geprüft haben, um welchen Fahren-Befehl es sich handelt und dann den entsprechenden Befehl an den String angehängt haben, um den Befehl zusammenzusetzen. Hier haben wir zunächst wie im Vorlesungsbeispiel einen Stack verwendet, was sich aber als unnötig herausgestellt hat, weshalb wir auf einen StringBuilder umgestiegen sind. Bei den anderen Exit-Methoden sind wir ähnlich vorgegangen. Das gesamte Java-Projekt befindet sich in der angehängten Zip-Datei.

Beispielsausgabe

```
StaplerLexer lexer = new StaplerLexer(CharStreams.fromString("forward 120 up 14"));
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPKO> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.2\bin\java
.exe' '@C:\Users\Chris\AppData\Local\Temp\cp_2joosbmw\herhu5ndkc0dbrmm.argfile' 'StaplerToAst'
Program.toString() = "forward120up14"
```

Aufgabe 3

a)

Wir haben die statische Semantikanalyse zur Baummethode unserer Java-Klasse StaplerBuilder.java hinzugefügt. Die Semantikanalyse überprüft den vollen String und teilt ihn wieder in die einzelnen Befehle auf, bevor der Sinn der angegebenen Distanzwerte getestet wird. Sind diese gültig, wird der boolesche Wert true zurückgegeben und das Programm wird fortgesetzt. Wird false zurückgegeben, schlägt die Semantikanalyse fehl. In diesem Fall wird eine Custom Exception geworfen und in einer Liste im Builder gespeichert, am Ende werden die Fehler gesammelt zurückgegeben und es wird kein Programm zurückgegeben.

Beispielsausgabe

"forward 1200000 up 14"

```
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPKO> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.2\bin\java
.exe' '@C:\Users\Chris\AppData\Local\Temp\cp_2joosbmw\herhu5ndkc0dbrmm.argfile' 'StaplerToAst'
[StaplerException: Ungültige distanz beim Fahren von: 1200000]
Program.toString() = "null"
```

Das vollständige Projekt finden Sie in der angehängten Zip-Datei.

b)

Wir haben einen Compiler für unsere Sprache erstellt, der den Staplercode in G-Code übersetzt, eine Sprache, die normalerweise zur Steuerung von CNC-Fräsen oder 3D-Druckern verwendet wird. Im Programm Compiler ruft der Visitor die Methode visitBefehl auf, die den entsprechenden Befehl in G-Code übersetzt. Hier ist ein Beispiel für die Eingabe:

```
Gcode:
YG1 X12;
Y0 Z14;
Y0 Z-20;
Gcode Ende
```

"forward 12 up 14 down 20"

>>

Für die Fehlerbehandlung lesen wir die statischen Fehler aus, nachdem der Builder fertig ist, und brechen den Kompilervorgang ab, wenn die Anzahl der Fehler größer als 0 ist.

```
if (builder.getNumberOfSemanticErrors() > 0){  
    System.err.printf(  
        "%d error(s) detected%n", builder.getNumberOfSemanticErrors());  
    System.exit( status: 1);  
}
```

Die Fehlerbeschreibung wurde bereits vom Builder geloggt, so dass nachvollzogen werden kann, warum der Kompilervorgang fehlgeschlagen ist.

Aufgabe 4

a)

Für die prozedurale Implementierung haben wir die Methoden `readLines`, `removeEmptyLines`, `removeShortLines`, `removeEmptyLines` und `totalLineLengths` implementiert.

Beim Aufruf von `readLines` wird eine Liste als Parameter übergeben, die dann über einen imperativen Zugriff schrittweise aus dem `BufferedReader` gefüllt wird.

In `removeEmptyLines` wird wieder mit der als Parameter übergebenen Liste gearbeitet.

Mit einer `for each`-Schleife wird über die Liste iteriert und alle leeren Zeilen werden entfernt.

Ähnlich wie `removeEmptyLines` funktioniert auch `removeShortLines`, nur dass hier nicht leere Zeilen entfernt werden, sondern alle Zeilen, die eine bestimmte Länge unterschreiten.

Auch beim Aufruf von `totalLineLengths` wird wieder die Liste übergeben. Somit werden alle Operationen des prozeduralen Programms auf einem Objekt ausgeführt, das in den verschiedenen Methoden manipuliert wird.

b)

Für die funktionale Implementierung haben wir die Eingabedatei als Stream eingelesen und führen alle Operationen auf diesem Stream aus, d.h. es werden keine Objekte manipuliert, sondern die Stream-Operationen werden immer auf die vorhergehende Operation angewendet, so dass die gesamte Logik in einer Zeile stattfinden kann und es keine Seiteneffekte wie bei der prozeduralen Implementierung mehr gibt.

```
int n = Files.lines(input) Stream<String>  
    .filter(s -> !s.isEmpty())  
    .filter(s -> s.length() >= MIN_LENGTH)  
    .mapToInt(String::length) IntStream  
    .sum();  
long stop = System.nanoTime();
```

c)

Bei der Zeitmessung haben wir stark schwankende Ergebnisse erhalten, aber nach mehreren Messungen ist die Tendenz erkennbar, dass die funktionale Implementierung etwas schneller ist als die prozedurale.

Hier die Durchschnittszeiten von 10 Messungen für 99645 Lines:

Procedural = 5747,9 microsec

Functional = 3795,6 microsec

Der Code für Aufgabe 4 ist ebenfalls in der angehängten Zip-Datei enthalten.

Beispielsausgabe


```
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPK0> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.2\bin\java
.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPK0\Aufgab
e4\target\classes' 'aufgabe4.TimingComparison'
Procedural = 99645 (20306 microsec)
Functional = 99645 (14776 microsec)
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPK0>
```

Aufgabe 5

Aufgabe 5 a)

Seite 25


1. $(X,Y,Z) = (\text{john}, \text{likes}, \text{fish}).$

 $(X,Y,Z) = (\text{john}, \text{likes}, \text{fish}).$

X = john,
Y = likes,
Z = fish

Einfaches Matching der Liste


2. $[\text{cat}] = [X|Y]$

 $[\text{cat}] = [X|Y].$

X = cat,
Y = []

X = cat ist Head der Liste Y ist der Tail in diesem Fall eine leere Liste


3. $[X,Y|Z] = [\text{mary}, \text{likes}, \text{wine}]$

 $[X,Y|Z] = [\text{mary}, \text{likes}, \text{wine}]$

X = mary,
Y = likes,
Z = [wine]

X eigenes Element Y ist der Head einer Liste und Z = [Wine ist der Tail]

4. $[[\text{the}, Y] | Z] = [[X, \text{hare}], [\text{is}, \text{here}]]$.

 $[[\text{the}, Y] | Z] = [[X, \text{hare}], [\text{is}, \text{here}]]$.


X = the,

Y = hare,

Z = [[is, here]]

X = the und Y = hare ist Head, Z ist der Tail


5. $[\text{golden} | T] = [\text{golden}, \text{norfolk}]$

 $[\text{golden} | T] = [\text{golden}, \text{norfolk}]$

T = [norfolk]

Tail T = [norfolk] ; golden verschwindet


6. $[\text{white}, \text{horse}] = [\text{horse}, X]$.

 $[\text{white}, \text{horse}] = [\text{horse}, X]$.

false

False da white = horse evaluiert wird

7. $[\text{white} | Q] = [P, \text{horse}]$

 $[\text{white} | Q] = [P, \text{horse}]$

P = white,

Q = [horse]

P = white ist Head Q ist Tail

Seite 26

fakultaet(0, 1).

fakultaet(N, Result) :-

N > 0,

N1 is N - 1,

fakultaet(N1, SubResult),

Result is N * SubResult.



Bei der Fakultät haben wir zuerst den Basisfall erarbeitet und haben dann rekursiv für N1 als Zählvariable die fakultaet Funktion erneut aufgerufen und das Ergebnis in SubResult gespeichert.

Seite 28

1. $\text{append}(X, Y, [1, 2, 3, 4])$.

Hier werden alle Elemente aus Y Schritt für Schritt in X geschrieben

2. $\text{append}(X, [1, 2, 3, 4], Y)$.

 <code>append(X, [1,2,3,4], Y).</code>	 <code>append(X, Y, [1,2,3,4]).</code>
<code>X = [],</code>	<code>X = [],</code>
<code>Y = [1, 2, 3, 4]</code>	<code>Y = [1, 2, 3, 4]</code>
<code>X = [_A],</code>	<code>X = [1],</code>
<code>Y = [_A, 1, 2, 3, 4]</code>	<code>Y = [2, 3, 4]</code>
<code>X = [_A, _B],</code>	<code>X = [1, 2],</code>
<code>Y = [_A, _B, 1, 2, 3, 4]</code>	<code>Y = [3, 4]</code>
<code>X = [_A, _B, _C],</code>	<code>X = [1, 2, 3],</code>
<code>Y = [_A, _B, _C, 1, 2, 3, 4]</code>	<code>Y = [4]</code>
<code>X = [_A, _B, _C, _D],</code>	<code>X = [1, 2, 3, 4],</code>
<code>Y = [_A, _B, _C, _D, 1, 2, 3, 4]</code>	<code>Y = []</code>
<code>X = [_A, _B, _C, _D, _E],</code>	
<code>Y = [_A, _B, _C, _D, _E, 1, 2, 3, 4]</code>	
<code>X = [_A, _B, _C, _D, _E, _F],</code>	
<code>Y = [_A, _B, _C, _D, _E, _F, 1, 2, 3, 4]</code>	

(2)

(1)

Bei dem Aufruf von 2 hängt sich das Programm irgendwann auf.

Prolog versucht, für X Werte zu finden, um die Verkettung zu ermöglichen.

Aufgabe 5 b)

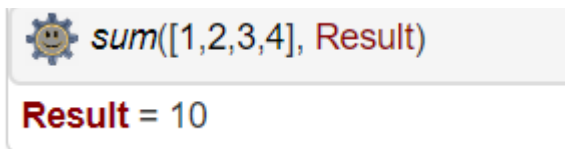
`sum([], 0). % Basisfall`

`sum(X, Result) :-`

`[H | T] = X,`

`sum(T, SubResult),`

`Result is H+SubResult.`



Aufgabe5 c)

Als erstes haben wir den Basisfall erarbeitet

`verbindung(Start, Startzeit, Ziel, Result) :-`

`zug(Start, X, Ziel, Y),`

`Startzeit < X,`

`Result = [Start, X, Ziel, Y].`

Dann haben wir den Rekursionsfall geschrieben

`verbindung(Start, Startzeit, Ziel, Result) :-`

`zug(Start, X, ZielZug, Y),`

`Startzeit < X,`

`Result = [[Start, X, ZielZug, Y] | Rest],`

`verbindung(ZielZug, Y, Ziel, Rest).`

Dazugehörige Bilder auf der folgenden Seite:

```

10 verbindung(Start, Startzeit, Ziel, Result) :-
11     zug(Start, X, Ziel, Y),
12     Startzeit < X,
13     Result = [Start, X, Ziel, Y].
14
15 verbindung(Start, Startzeit, Ziel, Result) :-
16     zug(Start, X, ZielZug, Y),
17     Startzeit < X,
18     Result = [[Start, X, ZielZug, Y] | Rest],
19     verbindung(ZielZug, Y, Ziel, Rest).

```

Beispielsausgabe

```

⚙ verbindung(konstanz, 08.00,mainz,Reiseplan)
Reiseplan = [[konstanz, 8.39, offenburg, 10.59], [offenburg, 11.27, mannheim, 12.24], [mannheim, 12.39, mainz, 13.18]]
Reiseplan = [[konstanz, 8.39, karlsruhe, 11.49], [karlsruhe, 12.06, mainz, 13.47]]
Reiseplan = [[konstanz, 8.53, singen, 9.26], [singen, 9.37, stuttgart, 11.32], [stuttgart, 11.51, mannheim, 12.28], [mannheim, 12.39, mainz, 13.18]]
false

```

Aufgabe 6

Für die sechste Aufgabe haben wir zunächst die Datei Stringtemplategroup aufgabe6.stg erstellt. Unsere Html-Seite wird durch mehrere Methoden zusammengesetzt. Von Java wird die Methode `surrounding(classes)` aufgerufen, der eine Liste von Klassen und Interfaces übergeben wird. Die Methode erstellt dann die Seitenstruktur der html Seite und den css style. Im Body der Html-Seite wird dann eine Methode aufgerufen, die unterscheidet, ob es sich bei den einzelnen Listenelementen um eine Java-Klasse oder ein Interface handelt.

```

dec(class) ::= <<
    $if (class.interface) $$interface(class)$$ else $$ class(class) $$endif$
>>

```

Wie im Codeausschnitt zu sehen, wird dann die entsprechende Methode aufgerufen, die die Tabelle für ein Interface oder eine Klasse erstellt.

Für ein Interface werden nur der Name des Interfaces und die Namen der Methoden, getrennt durch ein Trennzeichen, ausgegeben:

```

<td>
    $interface.methods; separator="<br>"$
</td>

```

Bei Interfaces wird für jedes Interface der Klasse eine weitere Methode aufgerufen, die dann den Namen des Interfaces und die zugehörigen Methoden in eine Zeile der Tabelle schreibt.

In der Java-Klasse, die das Template aufruft, wird zunächst aus allen übergebenen Klassen und Interfacenamen ein `Class<?>`-Objekt erzeugt und in eine Liste geschrieben. Dann wird ein String-Template erzeugt, die Liste der `Class<?>`-Objekte an das Template übergeben und das Ergebnis in eine HTML-Datei geschrieben.


```
STGroup group = new STGroupFile( fileName: "aufgabe6.stg");
ST template = group.getInstanceOf( name: "surrounding");
template.add( name: "classes", classes);
BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "test.html"));
writer.write(template.render());
writer.close();
```

Beispielsausgabe

Sprachkonzepte, Aufgabe 6

class java.lang.String:

Interface	Methods
java.io.Serializable	
java.lang.Comparable	public abstract int java.lang.Comparable.compareTo(java.lang.Object)
java.lang.CharSequence	public abstract int java.lang.CharSequence.length() public abstract java.lang.String java.lang.CharSequence.toString() public static int java.lang.CharSequence.compare(java.lang.CharSequence, java.lang.CharSequence) public abstract char java.lang.CharSequence.charAt(int) public default boolean java.lang.CharSequence.isEmpty() public default java.util.stream.IntStream java.lang.CharSequence.codePoints() public abstract java.lang.CharSequence java.lang.CharSequence.subSequence(int, int) public default java.util.stream.IntStream java.lang.CharSequence.chars()
java.lang.constant.Constable	public abstract java.util.Optional java.lang.constant.Constable.describeConstable()
java.lang.constant.ConstantDesc	public abstract java.lang.Object java.lang.constant.ConstantDesc.resolveConstantDesc(java.lang.invoke.MethodHandles\$Lookup) throws java.lang.reflectiveOperationException

interface java.util.Iterator:

Methods
public default void java.util.Iterator.remove() public default void java.util.Iterator.forEachRemaining(java.util.function.Consumer) public abstract boolean java.util.Iterator.hasNext() public abstract java.lang.Object java.util.Iterator.next()

class java.time.Month:

Interface	Methods
java.time.temporal.TemporalAccessor	public default int java.time.temporal.TemporalAccessor.get(java.time.temporal.TemporalField) public abstract long java.time.temporal.TemporalAccessor.getLong(java.time.temporal.TemporalField) public default java.lang.Object java.time.temporal.TemporalAccessor.query(java.time.temporal.TemporalQuery) public default java.time.temporal.ValueRange java.time.temporal.TemporalAccessor.range(java.time.temporal.TemporalField) public abstract boolean java.time.temporal.TemporalAccessor.isSupported(java.time.temporal.TemporalField)
java.time.temporal.TemporalAdjuster	public abstract java.time.temporal.Temporal java.time.temporal.TemporalAdjuster.adjustInto(java.time.temporal.Temporal)

Aufgabe 7

Wir haben für unsere Skriptsprache Python ausgewählt und holen uns von der API von brightsky das Wetter als JSON-File. In der URL geben wir die Koordinaten der HTWG mit, ebenso wie das aktuelle Datum, welches von Python automatisch eingefügt wird.

In der erhaltenen JSON-Struktur suchen wir den Eintrag der aktuellen Stunde, also den zuletzt gemessenen Wert, könnten aber auch in der Zukunft die Temperatur der kommenden Stunde vorhersagen, wenn wir bei dem Index das -1 weglassen würden. Je nach gewähltem Index können wir auch ganze Tage in die Prognose blicken.

Die aktuelle Temperatur wird dann auf die Konsole geprintet, zusammen mit dem Wetterstatus (also z.B. cloudy, sunny, etc.)

Hierbei verwenden wir in unserem Code folgende Aspekte die typisch für Skriptsprachen sind:

- Datentypen werden automatisch erkannt
- keine Typdeklaration und keine statische Typ- und Variablendeklaration
- keine Sichtbarkeiten
- keine main Methode, keine Klassen, keine Namespaces, keine Packages
- keine Kompilierung
- kompakteres Programm

Beispielsausgabe

```
PS D:\Users\Chris\OneDrive\Studium\Sprachkonzepte\SPK0> & C:/Users/Chris/AppData/Local/Programs/Python/Python312/python.exe d:/Users/Chris/OneDrive/Studium/Sprachkonzepte/SPK0/Aufgabe7/aufg7.py  
calling https://api.brightsky.dev/weather?lat=47.661942&lon=9.172430&date=2024-01-17  
Temperature was 0.6°C at 2024-01-17T10:00:00+00:00 while being rain
```

Der dazugehörige Python Code ist wieder im beigefügten Archiv verfügbar.

~~keine Interfaces~~

~~keine Abstrakten Klassen~~