Funktionale und objektorientierte Programmierkonzepte Übungsblatt 11

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Karsten Weihe

Übungsblattbetreuer:	Jonas Renk
Wintersemester 22/23	v1.0
Themen:	Streams
Relevante Foliensätze:	08
Abgabe der Hausübung:	27.01.2023 bis 23:50 Uhr

Hausübung 11 — Gesamt: 29 Punkte

L-Systeme mit Streams

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* im Moodle-Kurs.

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit `TODO` markierten `crash`-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h11` und ggf. `src/test/java/h11`.

Verbindliche Anforderungen für die gesamte Hausübung:

- In dieser Hausübung sind Ihn die Details ihrer Implementierung freigestellt. Allerdings ist die Benutzung von Schleifen und Rekursion (direkt oder indirekt durch Hilfskonstrukte) nicht erlaubt. Alle Aufgaben müssen über Streams gelöst werden.
- Nutzen Sie neben den der Java bereitgestellten Funktionalität immer die Ihnen zur Verfügung stehenden Attribute und erstellen Sie keine eigenen Instanzen, sofern dies nicht von der entsprechenden Aufgabenstellung explizit verlangt wird. Dementsprechend dürfen Sie im gesamten Übungsblatt davon ausgehen, dass keins dieser Attribute `null` ist.

Einleitung

In dieser Hausübung werden Sie sich mit L-System beschäftigen. L-Systeme sind ein Formalismus, mit dem Wachstumsprozesse beschrieben werden können. Bei einem L-System beginnt man mit einer Zeichenkette, z.B. A. Diese wird Axiom genannt. Weiter führt man Projektionen ein, um Elemente der Zeichenkette zu transformieren. Beispielsweise die Projektionen:

$$A \to AB,\ B \to A.$$

Mit diesen kann die oben Zeichenkette verändert werden, indem jedes Zeichen mit der entsprechenden Projektion ersetzt wird. Im folgenden wollen wir die linke Seite einer Projektion ihre „Quelle“ und die rechte Seite ihr „Ziel“ nennen.

Jede Zeile ist eine Iteration im L-System. Es werden immer alle Zeichen in der Zeichenkette ersetzt. Die Farben dienen als Hilfestellung, zu verdeutlichen, welche Projektionen genutzt wurden.

$$\begin{aligned}
A &&(n=0)\\
A &\to AB &(n=1)\\
AB &\to ABA &(n=2)\\
ABA &\to ABAAB &(n=3)\\
ABAAB &\to ABAABABA &(n=4)\\
ABAABABA &\to ABAABABAABAAB &(n=5)
\end{aligned}$$

Auf der rechten Seite jeweils die Zeichenkette nach n fachen projektieren. Dieses L-System wird „Algae“ genannt und kann dazu genutzt werden, die Fibonaccizahlen zu bestimmen.

Wenn Sie die Veranstaltung AfE, oder etwas Ähnliches, gehört haben, werden Sie hier den Begriff einer Grammatik wiedererkennen. Tatsächlich kann ein L-System als linkseindeutige, kontextfreie Grammatik beschrieben werden. Linkseindeutig deshalb, da sich zwei Projektionen nicht widersprechen dürfen. Für einen tieferen Einblick und einige weitere Beispiele können Sie einen Blick in den Wikipedia Artikel zu L-Systemen werfen. [1]

Um nicht auf Strings limitiert zu sein, werden Sie die L-Systeme als Liste implementieren und mit Streams transformieren. Auch wollen wir L-Systeme aus Dateien lesen können, damit wir diese nicht alle einzeln in Java abtippen müssen.

Schlussendlich werden Sie einige Testfälle für ihre Implementation schreiben. Die Testmethoden selbst werden Ihnen vorgegeben. Sie werden, wieder auf Streams aufbauend, die Testparameter generieren.

[1] `https://en.wikipedia.org/wiki/L-System`

H1: Algae 2 Punkte

L-Systeme werden durch das generische Interface `h11.LSystem` dargestellt. Der generische Parameter `T` repräsentiert den Typen, mit dem die Zeichen in den Zeichenketten des L-Systems kodiert sind.

Ist im Rest dieses Übungsblatts von einem „Zeichen“ die Rede, ist immer ein gültiger Wert des generischen Typparameters dieses Interfaces gemeint. Eine „Zeichenkette“ ist dementsprechend immer eine Sequenz solcher Werte.

In dem Interface sind zwei Methoden deklariert:

1. `getAxiom()`: Rückgabe dieser Methode ist `T`. Es soll immer das Axiom des jeweiligen L-Systems zurückgegeben werden. Im Beispiel aus der Einleitung wäre das A.

2. `project(T)`: Diese Methode repräsentiert die Projektionen des L-Systems. Der Parameter `e` vom formalen Typ `T` repräsentiert ein Zeichen aus dem L-System. Rückgabe ist die vom L-System zugewiesene Zeichenkette für das übergebene Zeichen. Im obigen Beispiel: `project(A)` $\rightarrow$ `[A, B]`. Die Zeichenkette wird hier als Stream kodiert.

Für eine erste Implementierung dieses Interfaces soll Ihnen die Klasse `h11.Algae` dienen. Neben den zuvor beschriebenen Methoden ist in ihr auch ein Enum `Variable` vorgegeben, das die Variablen aus der Einleitung repräsentiert.

H1.1: Das Axiom 1 Punkt

Implementieren Sie die Methode `getAxiom()` in der Klasse `h11.Algae`. Rückgabe ist das Zeichen A, gemäß des L-Systems.

H1.2: Die Projektionen 1 Punkt

In der Methode `project()` sollen Sie nun endlich die Projektionen des L-Systems implementieren. Geben Sie also je nachdem, ob Sie A oder B als Eingabe erhaben, AB oder B zurück. Jeweils wie zuvor beschrieben kodiert.

H2: Wachstum eines L-System 4 Punkte

Sie können nun L-Systeme beschreiben, aber noch nicht das Wachstum dieser berechnen. Damit sollen Sie sich jetzt beschäftigen. Implementieren Sie dafür in der Klasse `h11.LSystemGrowerImpl` die Methode `grow()`.

Die parameterlose Methode wird im Interface `h11.LSystemGrower` definiert und gibt einen Stream von Zeichenketten zurück. Genauer, einen unendlichen Stream von Listen von `T`, sodass für jede Liste L an Stelle $n \in \mathbb{N}$ gilt: L kodiert die Zeichenkette, die man erhält, wenn man das L-System n mal projektiert.

Es steht Ihnen ein Attribut vom Typ `LSystem` zur Verfügung, welches das L-System ist, das Sie projektieren sollen. Erstellen Sie zunächst eine Liste, die nur aus dem Axiom besteht. Dann rufen Sie `Steam.iterate` auf. Als erstes Argument übergeben Sie die erstelle Liste. Das zweite Argument ist ein Lambda-Ausdruck, der jedes Element mit dem Ergebnis des Aufrufs von `LSystem.project` ersetzt und aus diesem wieder eine Liste macht.

Hinweis:

Lesen Sie auch gerne nochmal im JavaDoc von `Stream`:

- `java.util.stream.Stream#iterate(T,java.util.function.UnaryOperator)`
- `java.util.stream.Stream#flatMap(java.util.function.Function)`

Unbewertete Verständnisfrage:

Warum müssen wir hier auf Listen zurückgreifen? Können wir keinen Stream von Stream nutzen? Probieren Sie es gerne aus, bedenken Sie aber, dass man einen Stream nur einmal durchlaufen kann.

H3: L-Systeme in Dateien speichern — 9 Punkte

Damit Sie nicht für jedes L-System eine eigene Klasse wie in H1 implementieren müssen, wollen wir nun für L-Systeme ein eigenes Dateiformat einführen und einen Parser dafür implementieren. Natürlich werden Sie auch dafür Streams nutzen. Zunächst ein Beispiel für das Dateiformat:

algae.ls

```
# Kommentare beginnen mit einem `#`
# Quelle der 1. Projektionen ist das Axiom
A -> AB
# Weitere Projektionen auf neuer Zeile
B -> A
```

Unter einem „Parser“ verstehen wir in diesem Fall eine Instanz einer Klasse, die es uns ermöglicht, das soeben beschriebe Textformat einzulesen. Dieses wird dann intern nicht mehr als Text, sondern über gewöhnliche Javaklassen dargestellt. Wenn Sie über den Begriff neugierig geworden sind, können Sie unter Anderem den Wikipedia Artikel zu Parsern überfliegen. [2]

Auch wenn wir bis jetzt L-Systeme generisch halten wollten, werden wir uns ab hier auf `java.lang.Character` als Zeichen einschränken. Alle fürs Parsen von L-Systemen relevante Klassen finden Sie im Paket `h11.parse`.

Um die geparsten Projektionen darstellen zu können, dient uns die Klasse `h11.parse.Projection`. Die Klasse hat zwei Attribute:

- `char source`: stellt das Zeichen, also die Quelle, einer Projektion dar.
- `String destination`: ist die Zeichenkette, auf die `source` abgebildet werden soll.

H3.1: Eingelesene L-Systeme — 3 Punkte

Als Repräsentanten für geparste L-Systeme dienen Ihnen die Instanzen der Klasse `h11.parse.ParsedLSystem`. Das Attribut `List<Projection> projections` ist eine Liste von Projektionen, die vom Parser eingelesen wurden. Die Klasse implementiert das Interface `LSystem`. Ihre Aufgabe ist das Implementieren der im Interface definierten Methoden.

In `getAxiom()` geben Sie einfach die Quelle der ersten Projektionen zurück. In `project(Character)` müssen Sie erst die Projektionen finden, deren Quelle mit dem Argument übereinstimmt. Dann geben Sie die Zeichenkette dieser

[2] `https://en.wikipedia.org/wiki/Parsing#Parser`

Projektion als Stream zurück. Wenn es keine solche Projektion gibt, geben Sie einfach einen Stream mit dem Argument als einziges Element zurück. Somit erlauben wir auch, dass Zeichen nicht weiter abgeleitet werden.

H3.2: Der Parser — 6 Punkte

Das Interface `h11.parse.LSystemParser` definiert eine Methode `parse` mit einem Parameter `lines` vom formalen Typ „`Stream` von String“. Dieser repräsentiert wie im obigen Beispiel die Zeilen. Rückgabe ist eine Liste, die jede eingelesene Projektionen beinhaltet. Sie sollen dieses Interface in der Klasse `h11.parse.LSystemParserImpl` implementieren. Gehen Sie in der Methode wie folgt vor:

1. Entfernen Sie die Kommentare aus jeder Zeile. Genauer: Wandeln Sie jede Zeile so um, dass alles nach dem ersten # entfernt wird. Zum Beispiel:

   `"A -> B # hello"` → `"A -> B "`

   `"# next line is funny"` → `""`

2. Löschen Sie alle Zeilen, die nur noch aus Leerzeichen bestehen.

3. Parsen Sie die verbleiben Zeilen und erstellen Sie daraus eine Instanz der Klasse `Projection`. Ist eine Zeile nicht im richtigen Format, werfen Sie eine `java.lang.IllegalArgumentException`.

4. Letztlich sammeln Sie alle eingelesen Projektionen in einer Liste.

Hinweis:

Im JavaDoc zu Strings finden Sie einige, nützliche Methoden:

- `java.lang.String#split(java.lang.String)`
- `java.lang.String#trim()`
- `java.lang.String#isBlank()`

H4: Testen der Algae — 5 Punkte

Sie haben nun ein L-System implementiert, das Fibonaccizahlen berechnet. Sie sollen jetzt Ihre Implementierung testen, indem Sie die Fibonaccizahlen zusätzlich auf einen herkömmlichen Weg berechnen und Ihre Ergebnisse vergleichen. Dafür werden Sie den Test im Paket `h11.fibs` implementieren.

H4.1: Generieren der Fibonaccizahlen — 2 Punkte

Für die Generierung implementieren Sie das Interface `FibonacciGenerator`, in dem eine Methode `generate` definiert ist. Der Parameter `numberOfFibs` vom formalen Typ `int` ist die Anzahl n und Rückgabe ist eine Liste der ersten n Fibonaccizahlen.

Achtung: In diesem Fall beginnen wir mit $1, 2$, nicht mit $0, 1$, wie es für die Fibonaccizahlen üblich ist. Es gilt also zum Beispiel `generate(5)` → `[1, 2, 3, 5, 8]`.

Implementieren Sie die Methode in der Klasse `FibonacciGeneratorImpl`, wobei Sie Folge $(a, b)_n$, mit

$$(a, b)_0 := (1, 2),$$
$$(a, b)_{n+1} := (b_n, b_n + a_n),$$

über `Stream.iterate` berechnen. Um ein Folgenglied $(a, b)_n$ zu speichern, können Sie Instanzen der Klasse `h11.fibs.FibonacciPair` nutzen.

Neben den Attributen `a` und `b` stehen Ihnen dort zwei Hilfsmethoden zur Verfügung. Einmal ein leerer Konstruktor, der die Attribute mit den Startwerten $1, 2$ initialisiert, sowie die parameterlose Methode `next`, die jedes Paar $(a, b)_n$ auf sein Nachfolger $(a, b)_{n+1}$ abbildet.

H4.2: Auslesen der Fibonaccizahlen aus der Algae — 2 Punkte

Implementieren Sie `FibonacciGenerator` gleich noch mal, dieses mal in der Klasse `AlgaeFibonacciGenerator`. Wieder sollen sie in `generate()` eine Liste der Fibonaccizahlen zurück geben.

Sie haben dazu ein Attribut `algaeGrower` vom Typ `LSystemGrower` zur Verfügung. Zur Erinnerung: Bei der Algae entspricht die Länge der Zeichenkette nach n-fachem Projektieren der n-ten Fibonaccizahl (Angefangen mit $1, 2$).

Gehen Sie wie folgt vor:

1. Erstellen Sie einen Stream der Algae
2. Begrenzen Sie seine Länge auf die Anzahl der Fibonaccizahlen
3. Weisen Sie jeder Zeichenkette ihre Länge zu
4. Machen Sie daraus wieder eine Liste

H4.3: Der Testcase — 1 Punkt

Nun können Sie endlich den Test schreiben. Implementieren Sie dafür in der Klasse `AlgaeTest` die Methode `testAlgaeGeneratesFibs` mit Parameter `numberOfFibs` vom Typ `int`. Sie haben Zugriff auf zwei Attribute `fibonacciGenerator` und `algaeFibonacciGenerator` vom Typ `FibonacciGenerator`, deren Werte Sie vergleichen sollen.

Der Test bekommt als Argument die Anzahl der Fibonaccizahlen. Generieren Sie zwei Listen dieser Länge und stellen Sie mittels Assertions sicher, dass das Attribut `algaeFibonacciGenerator` die richtigen Werte berechnet.

Sie dürfen in dieser Teilaufgabe davon aus gehen, dass das Attribut `fibonacciGenerator` korrekt arbeitet.

H5: Erweiterung von Random — 3 Punkte

Für die weiteren Aufgaben reicht uns die Klasse `java.util.Random` nicht ganz aus. Wir wollen Sie daher um zwei Methoden erweitern. Diese sind in der abstrakten Klasse `h11.AbstractRandom` definiert und sollen in der Klasse `h11.Random` implementieren werden.

Für diese Methoden bietet sich ein Blick ins JavaDoc der Methode `java.util.Random#ints(int,int)` an.

H5.1: Pythons `choices` — 1 Punkt

Implementieren Sie die Methode `choices()`. Der eine Parameter `values` vom formalen Typ Array von `T` ist eine

Sequenz an erlaubten Werten. Rückgabe soll ein Stream sein, sodass jedes Element im Stream mit gleicher Wahrscheinlichkeit ein Wert aus `values` sein kann.
Z.B: `choices(1, 2, 3)` → `[2, 2, 1, 3, 2, 1, 2, 3, 2, 2, ...]`

H5.2: Zufälliges Latein — 2 Punkte

Als Latein definierten wir das Alphabet $\{A, \ldots, Z\}$. Implementieren Sie die Methode `latin()`. Rückgabe ist ein String, bei dem jedes Element ein lateinisches Zeichen ist, und zwar gleich wahrscheinlich verteilt. Der Parameter `length` vom formalen Typ `int` definiert die Länge der Rückgabe.

> **Hinweis:**
>
> Schauen Sie sich nochmal die Darstellung von Zeichen in Java an: Kapitel 11, ab Folie 38 der FOP

H6: Zufällige L-Systeme — 4 Punkte

Damit Sie Ihren Parser möglichst abdeckend testen, sollen Sie L-Systeme per Zufall generieren. Hierzu implementieren Sie die Klasse `h11.providers.RandomLSystemGenerator`. Sie haben in den folgenden Methoden ein Attribut `random` zur Verfügung, dass die beschriebe Klasse `AbstractRandom` implementiert.

H6.1: Zufällige Projektionen — 1 Punkt

Implementieren Sie die Methode `makeProjection()`. Der Parameter `src` vom formalen Typ `String` soll die Quelle einer Projektion sein. Rückgabe soll eine Projektion mit der übergebenen Quelle und einem zufälligen Ziel sein. Das Ziel ist das Ergebnis von `random.latin()`, wobei das Argument wiederum eine Zufallszahl zwischen 1 und der Konstante `MAX_PROJECTION_DESTINATION_SIZE` ist.

H6.2: Zufälliges System — 3 Punkte

Implementieren Sie die parameterlose Methode `generate()`. Rückgabe ist eine Liste von Projektionen, wie sie im `ParsedLSystem` genutzt wird. Die Liste wird folgendermaßen erzeugt: Sei n eine Zufallszahl zwischen 1 und `MAX_SYSTEM_SIZE`. Generieren Sie einen Stream von n unterschiedlichen, zufälligen, lateinischen Strings der Länge 1. Wandeln Sie diese mit der Methode `makeProjection()` in Projektionen um und machen sie anschließend aus dem Stream eine Liste.

> **Hinweis:**
>
> `Stream` definiert hierfür zwei nützliche Methoden:
>
> - `java.util.stream.Stream#generate(java.util.function.Supplier)`
> - `java.util.stream.Stream#distinct()`

H7: Zufällige Darstellung eines L-Systems — 2 Punkte

In der vorherigen Aufgabe haben Sie L-Systeme erzeugt. Damit Sie jetzt auch Ihren Parser testen können, müssen Sie diese in das oben beschriebene Dateiformat übersetzten. Hier soll wieder der Zufall ins Spiel kommen.

Die Klasse `h11.providers.LSystemToRandomLinesConverter` ist dafür verantwortlich. Neben einem Attribut `random` vom Typ `AbstractRandom` haben Sie auch auf einige vorgegebene Methoden Zugriff.

H7.1: Zufällige Leerzeichen — 1 Punkt

Implementieren Sie die parameterlose Methode `generateSpaces()`. Rückgabe ist ein String der aus den Zeichen `' '` (Leerzeichen) und `'\t'` (Tabulator) besteht. Die Länge ist eine zufällige Zahl zwischen 0 und der Konstante `MAX_SPACES_SIZE`. Die Zeichen sollen gemäß `Random.choices` gleich verteilt sein.

H7.2: Darstellung für das ganze System — 1 Punkt

Implementieren Sie die Methode `lSystemAsLines()`. Die Methode bekommt einen Parameter `lSystem` vom formalen Typ Liste von `Projection` übergebenen. Die Liste ist wieder genau so aufgebaut, wie Sie in `ParsedLSystem` genutzt wird.

Als Rückgabe machen Sie aus `lSystem` einen Stream und rufen `flatMap` mit der Methode `projectionAsLines` als Argument auf.

H8: Ausführen der zufälligen Tests

Wenn Sie die Aufgaben H4 bis H7 gelöst haben, stehen Ihnen nun zusätzlich zu den Public Tests weitere Tests zur Verfügung, die Sie über den Gradle Task `verification/test` ausführen können.

Beachten Sie, dass diese Tests zufällige Eingabeparameter nutzen und daher schwer zu debuggen sein können. Sie können in der Klasse `h11.providers.ProjectionsProvider` das Attribut `seed` ändern, damit Ihre Testergebnisse reproduzierbar sind. Ist der Seed ungleich `null`, wird der von Ihnen gewählte Seed genutzt.