

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.3

Übungsblatt 0

Themen: Einführung, Informationen und Einstieg Racket

Relevante Folien: Orgafolien, Funktionale Abstraktion

Abgabe der Hausübung: 26.10.2018 bis 23:55 Uhr

Wichtiger Hinweis: Es ist immens wichtig, dass Sie sich intensiv und gewissenhaft mit den hier vorgestellten Themen beschäftigen, da sie die Grundlage für die Hausübungsaufgaben in diesem Semester darstellen. Auf dieser und der nächsten Seite erhalten Sie einige allgemeine Informationen, diese gelten für alle Hausübungen und wir setzen diese Informationen als gegeben voraus.

Aufbau der Übungsblätter

Die Übungsblätter gliedern sich immer in zwei Teile - die vorbereitenden Übungen **V** und die Hausübungen **H**. In den vorbereitenden Übungen sollen die Konzepte der Vorlesung eingebüttet werden. Die Aufgaben sind meist kürzer und sollen Sie praktisch an die Thematik heranführen. Außerdem bilden sie die Grundlage für die Hausübungen. Die Schwierigkeiten der einzelnen Übungen sind durch Sterne dargestellt (mindestens 0 und maximal 3), je mehr Sterne desto komplexer eine Aufgabe. Für ein sicheres Bestehen (das heißt 50% der Punkte) sollten Sie zumindest die Zwei-Sterne Aufgaben gut meistern können und die Drei-Sterne Aufgaben spätestens nach kleineren Hilfestellungen. Die Hausübungen sind meist etwas schwieriger als die vorbereitenden Übungen und werden am Ende bewertet. Zu den vorbereitenden Übungen wird es **keinen Lösungsvorschlag** geben! Die Lösungen der Hausübungen stellen wir Ihnen nach der Abgabefrist als Code auf moodle bereit.

Es gibt außerdem zu jedem Übungsblatt eine korrespondierende Fingerübung! Für nähere Informationen dazu, werfen Sie einen Blick auf Fingerübung 0.

Programmieren mit Stift und Papier

Bei Fragen zu den Übungen bieten wir Ihnen unterschiedlichste Hilfsangebote. Beachten Sie jedoch, dass Sie in der Abschlussklausur keine Hilfsmittel zur Verfügung haben werden. Üben Sie also schon zu Beginn auch ohne Entwicklungsumgebung und nur mit Stift auf einem Blatt Papier zu programmieren. Sie lernen das Ganze aber nur wenn Sie sich selbst intensiv mit dem Thema auseinander setzen, nutzen Sie Hilfe also nur wenn es gar nicht anders geht. Nur aus eigenen Fehlern lernen Sie richtig!

Hilfsangebote

- Treffpunkt/Hörsaalübung: Der Treffpunkt findet jeweils zu angekündigten Daten nach der Dienstagsvorlesung im Audimax statt. Hier werden teilweise Übungen besprochen und Hilfestellungen zur Bearbeitung gegeben.
- Sprechstunden: Auf der Kursseite finden Sie eine Übersicht über die Sprechstunden, die angeboten werden. Diese können Sie aufsuchen und den Tutorinnen und Tutoren gezielt Fragen zu den Inhalten stellen.
- Forum: In moodle sind Foren für die Übungsblätter eingerichtet worden. Stellen Sie dort Ihre Fragen mit einem möglichst aussagekräftigen Titel, so können auch alle anderen Kursteilnehmer maximal davon profitieren.

Abgabe der Hausübungen

Zur Hausübungsabgabe nutzen wir die entsprechenden Module in moodle. Die Abgabefristen sind auf jedem der Blätter vermerkt, sie liegen jedoch immer Freitagabends um 23:55 Uhr Serverzeit. **Es werden nur Bearbeitungen bewertet, die auf moodle hochgeladen wurden** (keine Repository-Links, keine Dropbox, keine E-Mails etc.). Sollte Ihnen beim Upload der Status "verspätet" angezeigt werden ist dies kein Problem. Solange Sie etwas auf moodle hochladen können, akzeptieren wir es noch. Halten Sie sich bei der Abgabe der Dateien an die folgenden von uns gegebenen Konventionen! Die Konventionen für Racket-Dateien finden Sie in dieser Hausübung, die für Java-Dateien in Hausübung 3. Bei allen Abgaben handelt es sich um **Einzelabgaben!** Auch wenn sie zusammenarbeiten, müssen sie unterschiedliche und eigen bearbeitete Lösungen abgeben!

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Sollten wir ein Plagiat entdecken, wird dies entsprechend von uns geahndet.

https://www.informatik.tu-darmstadt.de/studium_fb20/im_studium/studienbuero/plagiarismus/index.de.jsp

Jeder der nachfolgenden Punkte ist vor allem für die FOP relevant, da sie Formen von Plagiarismus darstellen und damit verboten sind:

- Das Abschreiben von Lösungen von Hausübungen und Übernehmen von Lösungs-Code zu Programmieraufgaben.
- Das Übernehmen von fremderdachten Lösungsansätzen ohne korrekte Zitierung.
- Das Ausgeben eigener Hausübungslösungen vor dem Abgabetermin an andere.

Beachten Sie vor allem auch den letzten Punkt. Wenn Sie Ihre Lösung an jemand anderen weitergeben, so machen Sie sich selbst eines Plagiats schuldig und müssen ebenfalls mit Maßnahmen rechnen.

V Vorbereitende Übungen

V1 DrRacket installieren



Installieren Sie zu Beginn DrRacket von folgendem Link auf Ihrem Rechner:

<http://racket-lang.org/download/>

Alternativ können Sie DrRacket auch auf den Poolrechnern in S202 nutzen. In Abbildung 1 sehen Sie, wie die Oberfläche dann aussieht.

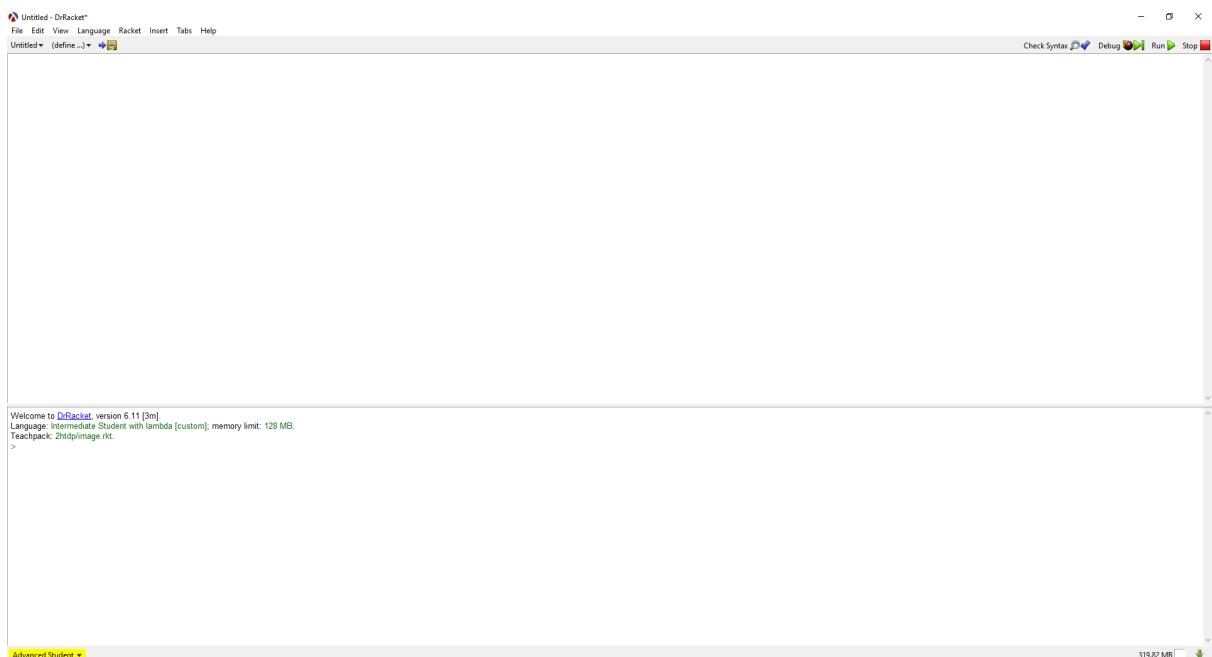


Abbildung 1: Oberfläche von DrRacket

Vergewissern Sie sich, dass links unten (im Screenshot gelb hinterlegt) auch *Advanced Student* steht. Andernfalls klicken Sie darauf und stellen es um.

Für erste Schritte in DrRacket, beachten Sie die dazugehörige Präsenzübung. Die Möglichkeiten der Entwicklungsumgebung wird auch ein Thema des ersten Treffpunkts sein.

V2 Arithmetik

Teil A: Ergänzen Sie folgende Ausdrücke durch Klammern, so dass jeweils ein gültiger Racket-Ausdruck entsteht. Werten Sie anschließend diese Ausdrücke aus. Gehen Sie für diese Aufgabe davon aus, dass alle Operatoren genau zwei Operanden erwarten.

1. $+ + + 7 3 4 * 2 3$
2. $* + 1 9 - 4 + 1 1$
3. $* - 0 * 2 3 - / 3 3 - 5 6$
4. $+ * 5 5 - - * 3 3 - * 4 4 4 4$

Teil B: Übersetzen Sie die folgenden mathematischen Ausdrücke in äquivalente Racket-Ausdrücke. Die Ausdrücke sollen nicht vereinfacht (gekürzt oder berechnet) werden. 4³ dürfen Sie durch eine entsprechende Multiplikation ersetzen. Um Ihre Eingaben zu überprüfen, haben wir Ihnen rechts das zu erwartende Ergebnis angegeben.

1. $\frac{13-3}{6 \cdot 3} \cdot (100 - 32)$ [= 37,7]
2. $(5 \bmod 3)/3.141 - 2$ [≈ -1.3632]
3. $\frac{4^3}{8} + (4 - 12)$ [= 0]

V3 Steuern

Zu jeder von Ihnen definierten Funktion wird ein Vertrag und mindestens 3 nichttriviale Tests erwartet. Mit nichttrivialen Tests sind insbesondere solche gemeint, die Randfälle abdecken, andererseits repräsentative “normale” Fälle! Nutzen Sie dafür `check-expect` und `check-within`. Vergleichen Sie dazu auch das Beispiel und die Erläuterung im Hausübungsabschnitt und in der Vorlesung. In dieser Aufgabe steht es nochmals explizit dabei, ab dem nächsten Blatt nicht mehr.

Die Steuern in Land X ergeben sich aus dem Einkommen multipliziert mit dem Steuersatz. Der Steuersatz ist wiederum 0,5% je tausend Euro Einkommen (abgerundet). So würde ein Einkommen von 40.000 Euro einem Steuersatz von $0,5\% \cdot 40 = 20\%$ und einer zu zahlenden Steuer von 8.000 Euro entsprechen, ein Einkommen von 23.700 Euro einem Steuersatz von 11,5% etc. Bei Einkommen größer als 200.000 Euro bleibt der Steuersatz bei 100%.

1. Definieren Sie eine Funktion `get-taxrate`, die ein Einkommen als Parameter erhält und den Steuersatz (in Prozent, d.h. 40 für 40%) zurückgibt. Nutzen Sie dazu die Funktion `floor`, die eine gegebene Zahl abrundet.
2. Definieren Sie eine Funktion `get-income`, die ein Einkommen übergeben bekommt und das Einkommen nach Abzug der Steuer zurückgibt.

H Nullte Hausübung

Gesamt 1 Punkt

Abgabe von Racket Hausübungen

Diese nullte Hausübung soll Sie mit den Abgabeformalitäten für die Racket Hausübungen vertraut machen. Für alle nachfolgenden Abgaben (in Racket) gilt:

Die hier vorgestellten Regeln müssen von Ihnen eingehalten werden! Dazu gehören zum einen die Vorgaben zum korrekten Abgabeprozedere, als auch die inhaltlichen Vorgaben von uns.

Für alle Hausübungen gibt es eine Vorlagendatei von uns in moodle. Nutzen Sie immer diese Vorlage und schreiben Sie keine eigene! Speichern Sie zunächst die heruntergeladene Vorlage an einem Ort Ihrer Wahl. Benennen Sie die Datei dann gleich korrekt um. Die Datei haben Sie dabei folgendermaßen zu benennen: Hnr_ln_fn. Dabei ist nr durch die aktuelle Übungsnummer zu ersetzen (bei einstelligen Zahlen ergänzen Sie eine führende 0), ln durch Ihren Nachnamen und fn durch ihren Vornamen. Max Mustermann gibt die Lösungsdatei für diese Übung also als **H00_Mustermann_Max.rkt** in moodle ab. Importieren Sie diese dann in DrRacket über *File* dann *Open* oder über den Shortcut Strg+O. Dort finden Sie bereits die Funktionen, die Sie in den Hausübungen zu definieren haben. Sie müssen die Funktionen also nur noch ergänzen. Möchten Sie Hilfsprozeduren definieren, so können Sie dies in der Vorlage einfach tun. Sie sind in der Anzahl der Hilfsfunktionen nicht beschränkt, achten Sie aber auf eine sinnvolle Zerlegung der Teilprobleme und lagern Sie nur dort aus, wo es auch sinnvoll ist.

Für **alle von Ihnen geschriebenen Funktionen** (egal ob vorgegebene Hauptfunktion oder neugeschriebene Hilfsfunktion) gilt:

Sie müssen Vertrag (Typ, Rückgabe und evtl. Precondition) und drei nichttriviale Tests für alle Funktionen angeben! Bei Lambda-Ausdrücken dürfen Sie auf die Tests verzichten, Vertrag ist aber auch hier verpflichtend. Orientieren Sie sich dabei an folgendem Beispiel, dass Ihnen schon aus der Vorlesung vertraut sein sollte (deutsch oder englisch ist egal):

```
;; Type: number number -> number
;; Returns: Euclidean norm of the vector (x,y)
(define (euclid2 x y)
  (sqrt (+ (* x x) (* y y))))
;; Tests
(check-expect (euclid2 3 4) 5)
(check-within (euclid2 2 2) 2.8284 0.001)
(check-within (euclid2 5.3 4.22) 6.774 0.001)
```

Achtung: Mit nichttrivialen Tests sind insbesondere solche gemeint, die Randfälle abdecken, andererseits repräsentative “normale” Fälle! Nutzen Sie dafür **check-expect** und **check-within**. Testen Sie also nicht nur mit trivialen Eingabedaten (im Beispiel oben bspws. Kombinationen aus 0 und 1).

Haben Sie die Hausübung bearbeitet und möchten Sie abgeben, so speichern Sie Ihre Ergebnisse und geben Sie diese als Datei mit .rkt Endung ab!

Wenn Sie eine der auf dem Blatt beschriebenen Anforderungen zur Abgabe und Dokumentation verletzen, so werden wir Ihnen Punkte von Ihrer Gesamtpunktzahl abziehen! Vermeiden Sie dies also, um nicht unnötig Punkte zu verlieren.

Laden Sie Ihre Datei dann rechtzeitig im Abgabemodul in moodle hoch. Sie können Ihre Abgabe innerhalb der Bearbeitungsfrist beliebig oft hochladen, die zuletzt hochgeladene Version wird bewertet.

Verbindliche Anforderung: Für alle Racket-Hausübungen gilt, dass Zuweisung mittels `set!`, `let`, `begin\textit{\textit{}}{}` oder ähnliches nicht erlaubt ist! Sollten Sie durch eigene Recherche auf dieses Konstrukt stoßen dürfen Sie es nicht verwenden, Sie müssen die Hausübungen mittels normaler Rekursion, wie in der Vorlesung vorgestellt, lösen.

H1 Richten Dateinamen bilden

1 Punkt

Laden Sie sich die Vorlage dieser Hausübung herunter.

Ergänzen Sie dort die Funktion `generate-filename`. Diese bekommt die Zahl als `nr`, welche die aktuelle Blattnummer angibt, und zwei Strings `ln` und `fn`, die den Vor- beziehungsweise Nachnamen angeben. Die Funktion gibt die korrekte Dateibenennung für die entsprechende Hausübung zurück.

Beachten Sie nochmals die Erläuterungen zur korrekten Benennung vor der Aufgabe.

Vergessen Sie nicht Vertrag und mindestens drei Tests!

Diese Aufgabe ist sehr einfach, sie dient hauptsächlich dazu, Sie mit den Abgabeformalitäten vertraut zu machen. Sie erhalten diesen Punkt deswegen auch nur, wenn wirklich **alles** eingehalten und beachtet wurde. Verschenken Sie diesen Punkt also nicht, so einfach können Sie dieses Semester keinen mehr ergattern.

Hinweise:

- Vergessen Sie die Dateiendung ".rkt" nicht!
- Mittels `string-append` können Sie beliebig viele Strings zusammenfügen.
Beispiel: (`string-append "Hallo " "Welt" "!"`) gibt "Hallo Welt!" zurück.
- Mittels `number->string` können Sie eine Zahl in einen String konvertieren.
Beispiel: (`number->string 42`) gibt "42" zurück.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.2

Übungsblatt 1

Themen: Grundlagen, Listen und Strukturen

Relevante Folien: Funktionale Abstraktion

Abgabe der Hausübung: 02.11.2018 bis 23:55 Uhr

V Vorbereitende Übungen

Denken Sie an Vertrag und drei Tests bei **jeder** Funktion.

V1 Temperaturumrechnung

★ ★ ★

Im angloamerikanischen Maßsystem wird die Temperatur nicht wie hierzulande in Grad Celsius gemessen, sondern in Grad Fahrenheit. Da Sie damit nichts anfangen können, wollen Sie sich nun eine Funktion `fahr->cel` definieren, welche die aktuelle Temperatur in Grad Fahrenheit übergeben bekommt und in Grad Celsius umwandelt.

Hinweis: Für eine gegebene Temperatur T_F in Grad Fahrenheit berechnet sich die dazugehörige Temperatur T_C in Grad Celsius über den folgenden Zusammenhang:

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

V2 Volumen eines Tetraeders

★ ★ ★

Das Tetraeder ist ein Körper mit vier dreieckigen Seitenflächen. Sein Volumen berechnet sich über die Formel $V(a) = \frac{\sqrt{2}}{12}a^3$, wobei a hier die Länge einer Kante ist. Sie sollen in dieser Aufgabe nun eine Funktion `tetrahedron-volume` schreiben, die für eine übergebene Kantenlänge a das Volumen des dazugehörigen Tetraeders zurückgibt. Gehen Sie dazu in folgenden Schritten vor:

1. Definieren Sie eine Funktion `pow3`, welche einen Parameter x bekommt, und den Wert x^3 zurückgibt. Erstellen Sie außerdem eine Konstante k , mit dem Wert $\frac{\sqrt{2}}{12}$.
2. Nutzen Sie die zwei vorherigen Schritte, um nun die Funktion `tetrahedron-volume` zu definieren, welche nur einen Parameter a bekommt.

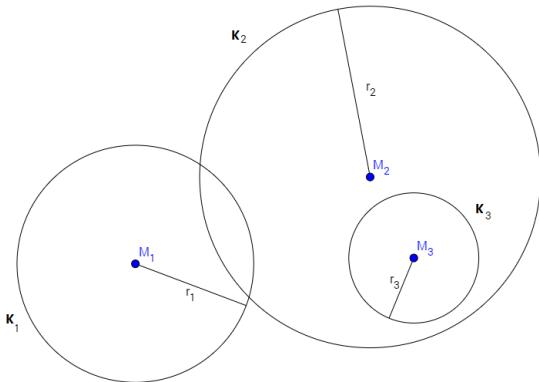
V3 Relative Lage zweier Kreise zueinander

★ ☆ ☆

Wir wollen die Lage zweier Kreise zueinander bestimmen. Definieren Sie dazu eine Prozedur `circles-position`, welche die Zahlen $x_1, y_1, r_1, x_2, y_2, r_2$ in dieser Reihenfolge entgegennimmt und einen String zurückgibt, welche die Lage der beiden Kreise zueinander beschreibt. Dabei sind x und y die Koordinaten eines Kreismittelpunktes und r der Radius des Kreises.

Zurückgegeben werden soll "Intersect" bei einem Schnitt der beiden Kreise, "External" bei keinerlei Überlappung oder "Interior" wenn einer der beiden Kreise vollständig im anderen liegt. Eine Berührung der beiden Kreise, egal ob von innen oder von außen, soll als Schnitt der beiden Kreise erkannt werden.

Zum besseren Verständnis finden Sie im Anschluss an die Aufgabe ein Beispiel und die mathematische Konkretisierung des Sachverhalts (dabei steht d für den Abstand der Mittelpunkte). Bei den Kreisen K_1 und K_2 im Beispiel sollte "Intersect", bei den Kreisen K_1 und K_3 sollte "External" und bei den Kreisen K_2 und K_3 sollte "Interior" zurückgegeben werden.



$$\text{circles-position} = \begin{cases} \text{Interior} & \text{if } d < |r_1 - r_2| \\ \text{External} & \text{if } r_1 + r_2 < d \\ \text{Intersect} & \text{else} \end{cases}$$

V4 Euklidischer Algorithmus

★ ☆ ☆

In der folgenden Aufgabe soll das Konzept der Rekursion verinnerlicht werden. Dazu werfen wir einen Blick auf eine Version des euklidischen Algorithmus, welcher Ihnen vielleicht schon aus der Mathematik bekannt ist. Für zwei natürliche Zahlen a und b lässt sich mit ihm der größte gemeinsame Teiler der beiden Zahlen berechnen. Dabei geht der Algorithmus wie folgt vor:

Gilt $b = 0$ so wird a zurückgegeben, ist hingegen $a = 0$ so wird b zurückgegeben. Gilt $a > b$ so wird der Algorithmus mit einem neuen $\hat{a} = a - b$ und dem "alten" b aufgerufen. Im anderen Fall wird der Algorithmus mit dem "alten" a und einem neuen $\hat{b} = b - a$ aufgerufen. Definieren Sie eine Prozedur (`euclid a b`), welche diese Version des euklidischen Algorithmus rekursiv umsetzt.

V5 Listenausdrücke auswerten

Teil A: Werden die folgenden Ausdrücke ohne Fehler durch Racket ausgeführt?

1. (cons 1 (cons 2 (cons 3)))
2. (cons 1 (list 2 (list (list 3 + 4))))
3. (list (cons empty 1)(cons 2 empty)(cons 3 empty))
4. (first empty)

Teil B: Liefern die folgenden Listenausdrücke dasselbe Ergebnis zurück?

1. (cons 1 (cons 2 (cons 3 empty))) und (list 1 2 3 empty)
2. (cons (list "(list)")empty) und (list "list" empty)
3. (list 7 "*" 6 "=" 42) und (cons 7 (cons "*" (cons 6 (cons "=" (list 42))))))

Teil C: Gehen Sie nun von folgendem Codeschnipsel aus:

```
(define A (list (cons 1 empty)(list 7) 9 ))
(define B (cons 42 (list "Hello" "world" "!" )))
```

Was liefern die folgenden Aufrufe zurück?

1. (first (rest A))
2. (rest (first A))
3. (append (first B)(rest (rest A))(first A))

V6 Strukturausdrücke auswerten

Gegeben sei folgende Strukturdefinition:

```
(define-struct my-pair (one two))
```

Was liefern die folgenden Aufrufe zurück?

1. (my-pair? (make-my-pair "a" "b"))
2. (make-my-pair 1 (make-my-pair 2 empty))
3. (* (my-pair-two (make-my-pair 1 2))(my-pair-one (make-my-pair 3 4)))

V7 Listen in Strukturen

Gegeben ist ein Struct-Typ abc mit zwei Feldern a und b. Definieren Sie eine Funktion foo mit einem Parameter p. Falls p vom Typ abc und zudem der Wert im Feld b von p eine Liste ist, liefert foo eine Liste zurück, deren erstes Element der Wert von Feld a in p ist, und der Rest der zurückgelieferten Liste ist die Liste im Feld b von p (also eine Liste in der Liste). Andernfalls liefert foo einfach false zurück.

V8 Suche in Zahlenliste

★ ★ ☆

Definieren Sie eine Funktion `contains-x?`. Diese bekommt eine Liste und eine Zahl übergeben und liefert genau dann `true` zurück, wenn die übergebene Zahl mindestens einmal in der übergebenen Liste vorkommt.

V9 Duplikate in Zahlenliste

★ ★ ☆

Definieren Sie eine Prozedur `duplicates?` mit einem Parameter `lst`, die genau dann `true` zurückliefert, wenn eine Zahl mehr als einmal in `lst` vorkommt. *Hinweis:* Können Sie hier vielleicht Ihre Funktion aus Aufgabe V8 verwenden?

V10 Verschachtelte Listen

★ ★ ★

Definieren Sie eine Prozedur `duplicates-deep?` mit einem Parameter `deep-lst`. Es wird erwartet, dass `deep-lst` entweder eine Zahl oder eine Liste ist, deren Elemente wiederum Zahlen oder Listen sind usw. (Liste von Listen von Zahlen). Die Prozedur `duplicates-deep?` soll `true` oder `false` zurückliefern, und zwar `true` genau dann, wenn mindestens eine Zahl mehr als einmal vorkommt.

Hinweise: Schreiben Sie sich eine Hilfsmethode `collect` mit zwei Parametern `lst` und `oracle`. Der Sinn von `oracle` ist es dabei, alle bereits vorgekommenen Zahlen abzuspeichern. Machen Sie also aus der verschachtelten Liste wieder eine normale Liste mithilfe von `collect`. Die Funktion aus V9 kann Ihnen hier sehr hilfreich sein.

V11 Arithmetisches Mittel

★ ★ ★

Gegeben seien folgende Strukturdefinition:

```
(define-struct person (age sex))
(define-struct student (person id))
```

Eine Person hat ein Alter (als Zahl) und ein Geschlecht (als String). Ein Student wiederum besteht aus einer Person (vorheriges Struct) und einer Matrikelnummer (ein String).

Definieren Sie nun eine Funktion `mean-of-ages`. Diese bekommt eine Liste von Studenten übergeben und gibt das arithmetische Mittel ihrer Alter zurück.

Hinweis: Das arithmetische Mittel \bar{x} berechnen Sie für n Alter x_1, \dots, x_n über:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{x_1 + \dots + x_n}{n}$$

H Erste Hausübung

Binärbäume

Gesamt 6 Punkte

Sie haben in Racket bereits die Datenstruktur *Liste* kennengelernt. In dieser ersten Hausübung betrachten Sie eine neue Datenstruktur - den *Binärbaum*. Nachfolgend wird es nur um Binärbäume mit der sogenannten Suchbaumeigenschaft gehen.

Werfen Sie einen Blick auf die Liste (`list 5 9 3 11 6 4 1`). In Abbildung 1 wird diese Liste mithilfe eines Binärbaums dargestellt.

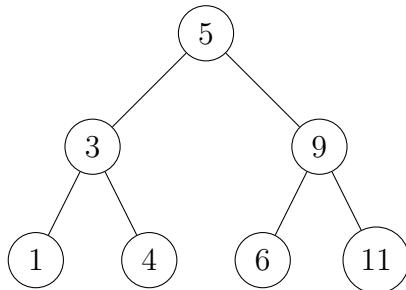


Abbildung 1: Ein Beispielbinärbaum mit Suchbaumeigenschaft

Der Baum besteht aus mehreren Teilen:

- Knoten des Baumes. Diese haben alle genau einen Vorgängerknoten und mindestens einen Nachfolgeknoten (das heißt einer der Nachfolgeknoten kann auch `empty` sein). In unserem Beispiel: 3,9.
- Wurzel des Baumes. Der Knoten ohne Vorgänger und damit zentrales Element des Baumes. In unserem Beispiel: 5.
- Blätter des Baumes. Die Knoten ohne Nachfolger. In unserem Beispiel: 1,4,6,11.

Die Suchbaumeigenschaft ist die zentrale Eigenschaft. Sie sagt aus, dass in den linken Nachfolgern eines Knotens nur Werte stehen, die kleiner als der aktuelle Knoten sind. In den rechten Nachfolgeknoten stehen nur größere Werte. Machen Sie sich dies an dem Beispiel oben klar. Betrachten wir beispielhaft den Knoten mit dem Wert 9 so steht links der kleinere Nachfolgewert ($6 < 9$) und rechts der größere ($11 > 9$).

In der Hausübungsvorlage finden Sie bereits eine gegebene Definition der Baumknoten:

```
(define-struct binary-tree-node (value left right))
```

Jeder Knoten wird also als Struct realisiert, welches einen Wert `value` besitzt und außerdem die beiden Nachfolgeknoten `left` und `right`. Bei den Blättern des Baumes setzen wir die Nachfolgerknoten auf `empty`.

Sie werden in den nachfolgenden Teilaufgaben exakt angeleitet und begleitet. Dabei gehen wir Schritt für Schritt vor.

Für ein besseres Verständnis können Sie einen Blick auf diesen Binary Search Tree Visualizer werfen.

H1 Suche im Binärbaum

2 Punkte

Zu Beginn sollen Sie sich mit der wichtigsten Eigenschaft des Binärbaums auseinandersetzen, der Suche von Werten im Baum. Machen Sie sich nochmal bewusst, wie die Suche in einer gewöhnlichen Liste abläuft (vergleichen Sie dazu Aufgabe V8). Sie müssen hier jedes Element überprüfen und die Funktion immer wieder rekursiv auf dem Rest der Liste aufrufen. Im schlimmsten Fall müssen Sie die Funktion solange rekursiv aufrufen, bis Sie ganz am Ende der Liste angelangt sind. Sie benötigen in diesen *Worst Case* Fällen ganze n Vergleiche bei einer n -elementigen Eingabeliste. Bei einem Binärbaum mit Suchbaum-eigenschaft werden hingegen deutlich weniger Aufrufe benötigt (näheres dazu lernen Sie in *Algorithmen und Datenstrukturen*).

Ergänzen Sie in dieser Aufgabe die Funktion `binary-tree-contains?`. Diese bekommt einen Wert als ersten und einen Binärbaum als zweiten Parameter übergeben. Die Funktion soll genau dann `true` zurückliefern, wenn der übergebene Wert im Binärbaum enthalten ist. Ist der übergebene Baum `empty`, so soll immer `false` zurückgegeben werden.

Hinweise:

- Der Binärbaum wird als **binary-tree-node**-Struct übergeben. Genauer gesagt wird die Wurzel des Baumes übergeben. Durch die Definition ihrer Nachfolger entstehen so verschachtelte Structs und insgesamt die Baumstruktur.
 - Nutzen Sie die Suchbaumeigenschaft! Wenn Sie die Knoten mit dem übergebenen Wert vergleichen, können Sie leicht entscheiden, welchen Nachfolgeknoten Sie besuchen sollten.
 - Überlegen Sie sich, wann Sie die Suche abbrechen. An welcher Stelle wissen Sie, dass der Baum den Wert nicht enthält?

In Abbildung 2 ist die (erfolglose) Suche nach dem Schlüsselwert 2 demonstriert. Der gelbe Knoten ist der aktuell besuchte, die roten Knoten sind bereits besuchte Knoten aus vergangenen Schritten. Machen Sie sich auch an diesem Beispiel nochmal klar, wie die Suchbaumeigenschaft funktioniert und wie Sie diese hier effektiv nutzen können.

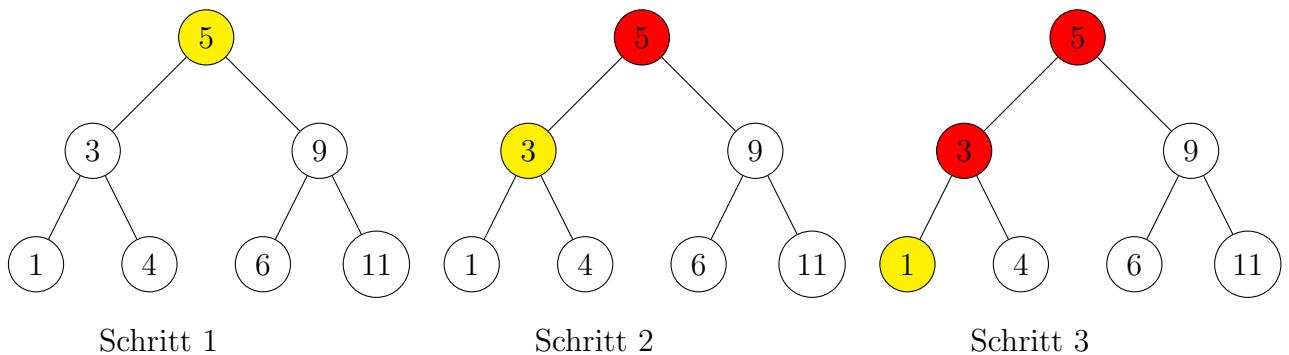


Abbildung 2: Erfolgslose Suche nach dem Schlüsselwert 2

In der Vorlage sind bereits Binärbäume für Sie von uns angelegt worden. Sie können diese verwenden, um Ihre Implementierung zu testen. Außerdem sind schon von uns Testfälle vorgegeben. Wenn Sie diese nutzen wollen, kommentieren Sie diese einfach wieder ein.

H2 Einfügen in den Binärbaum**2 Punkte**

Nun wollen wir neue Knoten in einen Binärbaum einfügen. Auch hier ist es wieder entscheidend, die Suchbaumeigenschaft auszunutzen. Als Beispiel nehmen wir wieder unseren altbekannten Baum und wollen den Wert 7 neu einfügen. In Abbildung 3 sind alle Knoten gelb markiert, die besucht werden, um die richtige Einfügeposition zu finden.

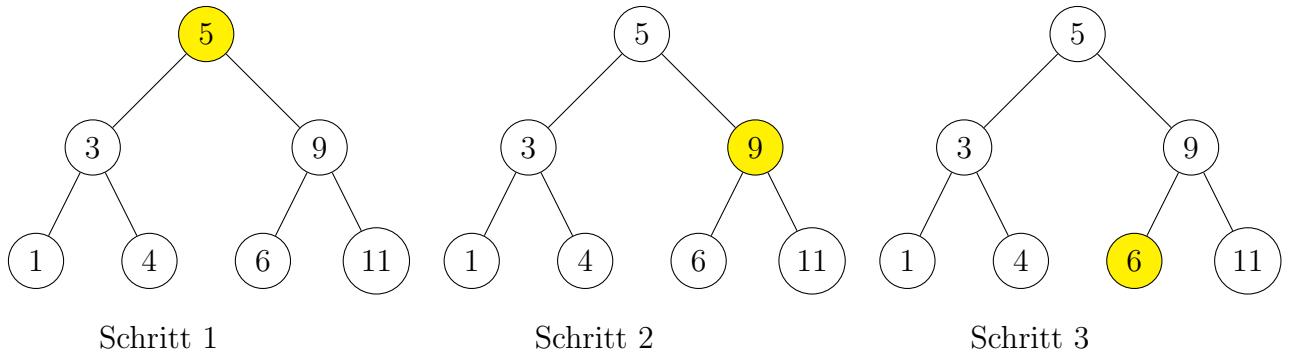


Abbildung 3: Finden der Position für den neuen Knoten mit Wert 7

Haben wir die richtige Stelle gefunden, so wird der neue Knoten eingefügt - hier in Abbildung 4 wieder gelb markiert.

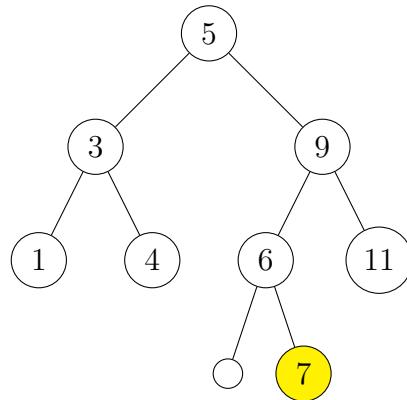


Abbildung 4: Der resultierende Baum nach dem Einfügen

Ergänzen Sie die Funktion `insert-into-binary-tree` in der Codevorlage. Diese erhält einen Wert und einen Binärbaum - wieder als Wurzelknoten-Struct - und soll diesen Wert korrekt einfügen. Soll ein Wert eingefügt werden, der bereits existiert, so soll der übergebene Baum ohne Modifizierung zurückgegeben werden.

Hinweis: Beachten Sie, dass der übergebene Binärbaum auch `empty` sein kann. In diesem Fall erstellen Sie nur die Wurzel des Baumes.

H3 Binärbaum aus Liste**1 Punkt**

In dieser Aufgabe sollen Sie eine vorhandene Liste in die Binärbaum Datenstruktur umwandeln. Ergänzen Sie dazu die Funktion `binary-tree-from-list`, welche eine Liste von Zahlen übergeben bekommt. Die Funktion soll den aus der Liste resultierenden Baum zurückgeben, wenn die Zahlen der Liste in der gleichen Reihenfolge in einen leeren Binärbaum eingefügt werden. Ist die übergebene Liste leer, so soll die Funktion `false` zurückgeben.

Hinweis: Überlegen Sie sich, wie Sie bei dieser Aufgabe sinnvoll eine Hilfsmethode verwenden könnten.

H4 Duplikate im Binärbaum**1 Punkt**

Im Normalfall kommen im Binärbaum keine Werte mehrfach vor. In dieser Aufgabe sollen Sie jedoch eine mögliche Behandlung für solche Duplikate implementieren. Ergänzen Sie die Funktion `insert-into-binary-tree-duplicates`. Diese bekommt wieder einen Binärbaum und einen Wert übergeben, der eingefügt werden soll. Die Besonderheit dieser Methode: Ist der Wert bereits im Binärbaum vorhanden, so soll das Feld `value` des `binary-tree-node`-Structs durch eine Liste ersetzt werden, in der der Wert n-mal vorkommt (bei n-maligem Einfügen in den Baum). Wird beispielsweise der Wert 7 jetzt 5 mal in den Baum eingefügt werden und befindet sich der Wert 7 an einem Blatt, so sieht das dazugehörige Struct folgendermaßen aus:

```
(make-binary-tree-node (list 7 7 7 7 7) empty empty)
```

Hinweis: Sie können Ihre Funktion `insert-into-binary-tree` aus Übung H2 wiederverwenden und entsprechend modifizieren.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.0

Übungsblatt 10

Themen: Wrap-Up I

Relevante Folien: alle bis dahin

Abgabe der Hausübung: 25.01.2019 bis 23:55 Uhr

Dies ist eines der drei Wrap-Up Blätter. Auf diesen Blättern begegnen Ihnen keine völlig neuen Konzepte mehr, sondern sie stellen vielmehr einen Rundumschlag über alle vergangenen Themen und spezielle Vertiefungen dar.

Die vorbereitenden Übungen enthalten auf diesen Blättern keine Sternkennzeichnung mehr.

Außerdem sind die Hausübungen etwas kniffliger als die bisherigen. Auf diesem fortgeschrittenen Level müssen Sie sich davon verabschieden, dass wir Ihnen genau mittels Teilaufgaben strukturieren, wie Sie ein Problem zu lösen haben. Überlegen Sie sich selbst eine sinnvolle Strukturierung und lösen Sie die Aufgaben genau so, wie Sie sich das vorstellen.

V Vorbereitende Übungen

V1 Fibonacci: rekursiv vs. iterativ

Die Fibonacci Funktion ist eines der klassischen Beispiele für Rekursion. Sie ist wie folgt definiert:

$$\text{fib}(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ \text{fib}(n - 1) + \text{fib}(n - 2) & \text{falls } n \geq 2 \end{cases}$$

1. Implementieren Sie eine Methode in Java, welche die Fibonacci Funktion nach obiger Definition rekursiv berechnet.
2. Implementieren Sie nun erneut eine Methode in Java, welche die Fibonacci Funktion berechnet. Diesmal muss Ihre Lösung allerdings iterativ (also über Schleifen) gelöst werden!

V2 Ackermann Funktion

Die Ackermann-Funktion ist eine 1926 von Wilhelm Ackermann gefundene mathematische Funktion, die in der theoretischen Informatik eine wichtige Rolle bezüglich Komplexitätsgrenzen von Algorithmen spielt. Sie wird als Benchmark zur Überprüfung rekursiver Prozeduraufälle verwendet, wenn man die Leistungsfähigkeit von Programmiersprachen oder Compilern überprüfen will.

Gegeben sei folgende vereinfachte Version der Ackermann Funktion:

$$\begin{aligned}\text{Ack}(0, m) &= m + 1 \\ \text{Ack}(n, 0) &= \text{Ack}(n - 1, 1) \\ \text{Ack}(n, m) &= \text{Ack}(n - 1, \text{Ack}(n, m - 1))\end{aligned}$$

Implementieren Sie die Funktion in Racket.

1. Berechnen Sie den Aufruf `Ack(2,2)`.
2. Berechnen Sie die ersten Schritte des Aufrufs `Ack(4,2)`. Was fällt Ihnen hierbei sehr schnell auf?

V3 Pascal'sches Dreieck

Definieren Sie in Racket eine **rekursive** Funktion (`pascal-line n`), die die n -te Zeile des Pascal'schen Dreiecks berechnet. Jede Zeile n im Dreieck, beginnend mit Zeile 0, hat $n + 1$ Elemente.

Für die n -te Zeile und die $n + 1$ Elemente und die Positionen i pro Zeile gilt dann:

$$p_i(n) = p_{i-1}(n - 1) + p_i(n - 1) \quad \text{mit} \quad i = 1 \dots n + 1 \quad \text{und} \quad p_0(n - 1) = 0$$

Nachfolgend ist das Pascal'sche Dreieck mit den ersten 7 Zeilen angegeben. Jeder Wert in einer gegebenen Spalte entsteht durch die Summe des Wertes direkt darüber und links schräg darüber. Falls der Wert links darüber nicht existiert, ist der Wert 0 dafür anzunehmen.

$n = 0$					1			
$n = 1$				1		1		
$n = 2$			1		2		1	
$n = 3$		1		3		3		1
$n = 4$	1		4		6		4	1
$n = 5$	1	5		10		10	5	1
$n = 6$	1	6	15		20		15	6
								1

Ihre Funktion soll die n -te Zeile jeweils als Liste von Zahlen zurückliefern.

V4 Selection Sort

Bei dem Sortierverfahren *Selection Sort* wird beim Durchlaufen des Arrays der Größe n im Durchlauf $i = 0, \dots, n - 1$ das kleinste Element ab Position $i + 1$ bestimmt und mit dem

Element an Position i vertauscht. Daher verringert sich die Länge der zu sortierenden Arrayteilfolge mit jeder Iteration um eins, womit sichergestellt ist, dass der Algorithmus terminiert.

Implementieren Sie nun Selection Sort in Java als `void selectionSort(int[] array)`.

Beispiel für das Array [4,2,1,6,3,5]:

i = 0	[4,2,1,6,3,5]	4 ↔ 1
i = 1	[1,2,4,6,3,5]	-
i = 2	[1,2,4,6,3,5]	4 ↔ 3
i = 3	[1,2,3,6,4,5]	6 ↔ 4
i = 4	[1,2,3,4,6,5]	6 ↔ 5
i = 5	[1,2,3,4,5,6]	

Für weitere Informationen zum Algorithmus finden Sie beispielsweise in der deutschsprachigen Wikipedia unter:

<https://de.wikipedia.org/wiki/Selectionsort>

V5 String Matching

In dieser Aufgabe bekommen Sie in Racket zwei Strings und sollen prüfen, ob der eine String ein Teilstring des anderen ist, also ob der eine String im anderen enthalten ist. So ist beispielsweise "kuchen" Substring des Strings "Apfelkuchen".

Definieren Sie also eine Funktion (`(is-substring? s sub)`). Diese liefert `true` genau dann zurück, wenn der String `sub` ein Teilstring des Strings `s` ist.

Hinweis:

Sie können die eingebaute Funktion `explode` verwenden. Diese bekommt einen String übergeben und überführt diesen in eine Liste von einelementigen Strings.

Beispielsweise liefert (`explode "abc"`) die Liste (`list "a" "b" "c"`) zurück.

V6 Einführung Backtracking - Das n -Damen Problem

In der Hausübung des ersten Wrap-Up Blatts wollen wir uns mit sogenannten Backtracking-Algorithmen beschäftigen. Diese Aufgabe dient als erste Einführung, um das Prinzip dieser Algorithmen zu verinnerlichen.

Backtracking-Algorithmen

Es gibt Probleme, die wir mit der kennengelernten Rekursion aus der Vorlesung nicht oder zumindest nicht optimal gelöst bekommen. Dafür gibt es Backtracking-Algorithmen, die auf folgender Vorgehensweise basieren:

1. Verfolge einen möglichen Lösungsweg, bis ...
 - (a) ... die Lösung gefunden wurde: **Erfolg!**
 - (b) ... oder der Weg nicht fortgesetzt werden kann.
2. Wenn der Weg nicht fortgesetzt werden kann: Gehe den Weg zurück, bis zur letzten Verzweigungsmöglichkeit, wo es noch nicht gewählte Alternativen gibt und wähle dort noch eine nicht gewählte Alternative und mache weiter bei Schritt 1.
3. Wenn der Ausgangspunkt erreicht ist und es keine Alternativen mehr gibt: **Miss-Erfolg!**

V6.1 Das n -Damen Problem

Bei dem n -Damen Problem handelt es sich um ein mathematisches Problem aus der Welt des Schachs.

Problemstellung: Wir wollen n Damen auf einem $n \times n$ -Schachbrett so positionieren, dass keine Dame eine andere gemäß der Schachregeln schlagen kann. Anders ausgedrückt: Es dürfen keine zwei oder mehr Damen auf derselben Reihe, Linie oder Diagonale stehen.

Für weitere Details bietet sich der deutschsprachige Wikipedia Artikel <https://de.wikipedia.org/wiki/Damenproblem> an. Hier findet sich bereits die Formulierung eines Algorithmus, welcher die Lösung mittels rekursivem Backtracking ermittelt. Versuchen Sie es aber zunächst unbedingt alleine!

Aufgabe: Implementieren Sie in Java die Methode `int[][] solveNQueens(int n)`. Die Funktion soll für ein gegebenes n eine mögliche Lösung des Problems mittels rekursivem Backtracking bestimmen. Zurückgegeben wird ein $n \times n$ Array vom Typ `int`, wobei eine 0 bedeutet keine Dame und eine 1 bedeutet hier wurde eine Dame platziert.

Der Aufruf `solveQueens(4)` kann also beispielsweise folgendes Array zurückliefern:

0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0

H Zehnte Hausübung**Gesamt 11 Punkte*****Racket Reloaded: Backtracking***

In dieser zehnten Hausübung beschäftigen wir uns mit Backtracking (vergleiche V6) in Racket.

H1 Graphen traversieren**4 Punkte**

Als Graph bezeichnen wir eine Sammlung von Knoten und Kanten. In einem gerichteten Graphen repräsentieren die Kanten gerichtete Verbindungen zwischen den Knoten. Betrachten Sie das nachfolgende Beispiel eines solchen Graphens:

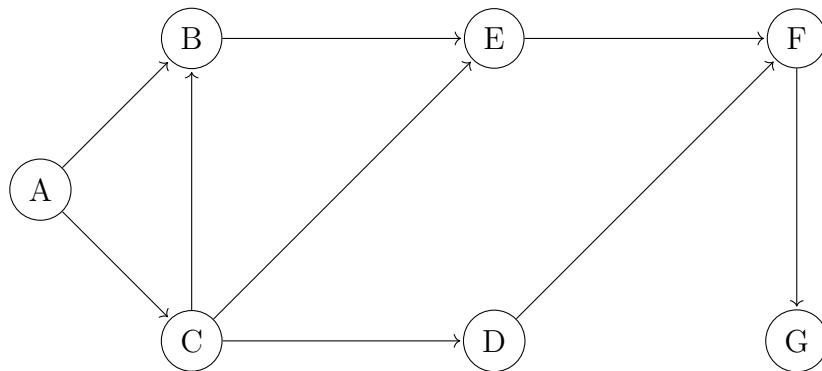


Abbildung 1: Beispielgraph

Sie dürfen im nachfolgenden davon ausgehen, dass wir nur gerichtete Graphen betrachten, welche keine Zyklen enthalten.

Wir wollen den eben gezeigten Graphen nun in Racket modellieren und verwenden dazu Strukturen und Listen. Die Knoten modellieren wir mit einem Namen (als String) und einer Liste mit den Namen seiner Nachbarn (Liste von Strings). Für unseren Beispielgraphen sieht dies wie folgt aus:

```

(define-struct node (name neighbors))
(define A (make-node "A" (list "B" "C")))
(define B (make-node "B" (list "E")))
(define C (make-node "C" (list "B" "D" "E")))
(define D (make-node "D" (list "F")))
(define E (make-node "E" (list "F")))
(define F (make-node "F" (list "G")))
(define G (make-node "G" empty))
  
```

Der Graph selbst besitzt ebenfalls einen Namen und eine Liste all seiner Knoten (als Liste von node-Structs). Für unseren Beispielgraphen sieht dies wie folgt aus:

```

(define-struct graph (name nodes))
(define G1 (make-graph "G1" (list A B C D E F G)))
  
```

Ihre Aufgabe:

Ergänzen Sie die folgende Funktion aus der Vorlage:

```
;; Type: node node graph -> (list of String)
;; Returns: a path from origin to dest in the given graph
;; Precondition: cycle-free graph
(define (find-route origin dest graph)
  ...
)
```

Beispielsweise liefert der Aufruf (`find-route B F G1`) die Liste (`list "B" "E" "F"`) zurück. Die Namen des Start- bzw. Zielknotens soll also auch nochmal in der Liste vorhanden sein.

Wird kein Pfad gefunden, so soll `false` zurückgegeben werden.

Wichtige Hinweise:

- In der Vorlage ist die Definition des Beispielgraphens gegeben, nutzen Sie diese um Ihre Implementierung zu testen.
- Nutzen Sie das vorgestellte Prinzip des Backtrackings!
- Sollte es mehrere Pfade geben, so ist es egal, welchen Pfad Sie zurückgeben. Die Funktion soll lediglich einen gültigen Pfad zurückliefern.

Beispiel für Backtracking:

Suchen wir beispielsweise den Pfad von A nach D, so sehen wir schnell, dass lediglich die Route (`list "A" "C" "D"`) in Frage kommt.

Nehmen wir an unser Algorithmus entscheidet sich nach C zunächst für den Knoten E. Er würde weiter nach F laufen und dann nach G und hätte keine Optionen mehr. Da wir nicht am Ziel sind gehen wir einen Knoten zurück und sind wieder bei F und schauen nach Alternativrouten, da es keine gibt gehen wir zurück nach E. Auch hier haben wir keine anderen Möglichkeiten, wir gehen also zurück nach C. Bei C haben wir zwei andere Alternativen und zwar B oder D. Entscheiden wir uns zunächst für B passiert das gleiche wie zuvor wir laufen zu G und dann wieder zurück zu C. Wählen wir jetzt D sind wir am Ziel und der Algorithmus terminiert mit einer korrekten Lösung.

H2 Einstellung neuer FOP-Tutoren**4 Punkte**

Wir stehen mal wieder vor einem Problem. Wir wollen neue FOP-Tutoren einstellen und haben lediglich ein begrenztes Budget. Unsere Entscheidung welche Tutoren wir nun einstellen, wollen wir dabei möglichst optimiert treffen. Tutoren besitzen zwei essentielle Eigenschaften: Schnelligkeit und Korrektheit. Beide Eigenschaften können wir auf einer Skala von 0-100 angeben, wobei höhere Werte besser sind. Jeder Tutor verlangt außerdem ein festes Gehalt für seine gesamte Arbeitszeit. Zur Modellierung verwenden wir folgende Struktur:

```
(define-struct tutor (name salary speed correctness))
```

Daraus ergibt sich für uns das Problem, dass wir nicht alle Tutoren anstellen können, wir müssen also eine Auswahl treffen!

Ihre Aufgabe:

Ergänzen Sie die folgende Funktion aus der Vorlage:

```
; ; Type: (list of tutor) number (tutor -> number) -> (list
; ; of String)
; ; Returns: the best combination of tutors that have a total
; ; price less than the number passed in, regarding the given
; ; criterion
(define (choose-tutors possible-tutors budget criterion)
  ... )
```

Die Funktion bekommt der Reihe nach die Liste der potenziellen Tutoren, das maximale zur Verfügung stehende Budget, sowie eine Auswertungsfunktion, welche angibt welches Kriterium wir maximieren möchten (hier also beispielsweise `tutor-speed` bzw. `tutor-correctness`) übergeben.

Geliefert werden soll eine Liste mit den Namen der Tutoren, für die die ausgewählte Auswertungsfunktion das Maximum, also die für unser Kriterium beste Zusammenstellung, liefert. Beachten Sie dabei unbedingt, dass das maximal verfügbare Budget nicht überschritten werden darf! Ihre Aufgabe ist es nun, entsprechend der konkret zu nutzenden Auswertungsfunktion die bestmögliche Auswahl an Tutoren zu treffen. Dabei können Sie also eine Tutorenliste mit entweder der maximalen Schnelligkeit, oder aber der maximalen Korrektheit erstellen.

Verbindliche Anforderung: Die Liste darf zu keinem Zeitpunkt sortiert werden. Ebenso darf nicht explizit die Potenzmenge berechnet und daraus dann die beste Lösung bestimmt werden. Beides ist für die Lösung auch nicht nötig. Sie dürfen also nicht einfach alle Lösungen bestimmen und dann die beste heraussuchen, dies wäre höchst ineffizient, verwenden Sie Backtracking!

Für diese Aufgabe sind bereits Beispieldaten und Tests in der Vorlage gegeben, Sie können auf die Erstellung von eigenen Tests bei dieser Aufgabe verzichten.

H3 Das n -Damen Problem - Reloaded**3 Punkte**

In allen Backtracking-Problemen bisher ging es darum, dass Sie **eine** mögliche Lösung finden. In dieser Aufgabe interessieren wir uns hingegen für die mögliche Anzahl **aller** Lösungen. Dazu betrachten wir erneut das n -Damen Problem aus Aufgabe V6.

Definieren Sie eine Funktion (`n-queens n`), welche wieder n Damen auf einem $n \times n$ großen Schachbrett verteilen soll. Die Funktion liefert die gesamte Anzahl an möglichen Lösungen zurück, nicht die Lösungen selbst! Lösungen die gespiegelte oder rotierte Varianten von anderen Lösungen sind zählen auch!

Beispiel: Für ein 8×8 Schachbrett gibt es nur 12 vollständig unterschiedliche Lösungen, allerdings insgesamt 92.

Unverbindliche Hilfestellung:

Sie können zur Lösung ein Struct `queen` verwenden:

```
(define-struct queen (x y))
```

Außerdem die folgende Funktion, welche testet ob eine Dame platziert werden kann:

```
; ; Type: queen (list of queen) -> boolean
; ; Returns: true if a queen can be attacked by a list of
; ; other queens
(define (under-attack new-queen queens)
  (cond
    [(empty? queens) false]
    [else
      (or
        ; ; queens on same row
        (= (queen-x new-queen) (queen-x (first queens)))
        ; ; queens on same col
        (= (queen-y new-queen) (queen-y (first queens)))
        ; ; queens on same diagonal
        (= (abs (- (queen-x new-queen) (queen-x (first queens)))) 
            (abs (- (queen-y new-queen) (queen-y (first queens))))))
        ; ; next queen
        (under-attack new-queen (rest queens))))]))
```

So ergibt `(under-attack (make-queen 1 1)(list (make-queen 1 2)))` beispielsweise `true`.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.3

Übungsblatt 11

Themen: GUI, Streams + Files

Relevante Folien: Threads + GUIs, Streams + Files

Abgabe der Hausübung: 01.02.2019 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Theoriefragen

☆ ☆ ☆

Welche der folgenden Aussagen ist wahr?

- (A) Streams können aus Listen und Arrays erzeugt werden.
- (B) Streams bieten keine Möglichkeit für elementweisen Zugriff.
- (C) Ein Objekt der Klasse Path verwaltet den Pfadnamen einer Datei oder eines Verzeichnisses oder eines anderen Objektes, das im jeweiligen Dateisystem über einen Pfadnamen ansprechbar ist.
- (D) Streams können in sich Daten speichern.
- (E) Byteweiser Zugriff ist nur dann sinnvoll, wenn eine Datei nur Text und keine Bilder, Audio- oder Videodateien enthält, da bei einem Text die Bytes leichter gelesen werden können.
- (F) Runnable ist ein Functional Interface. Die funktionale Methode heißt run, hat keine Parameter und ist void.
- (G) Einzelne Threads können nicht terminiert werden, sie enden alle mit dem Ende des Gesamtprogramms.
- (H) Eine Parallelisierung mit Threads verkürzt immer die Laufzeit eines Programms.
- (I) Wann immer auf einen Button geklickt wird, wird Methode `actionPerformed` jedes bei diesem Button registrierten Listeners aufgerufen.
- (J) Für jede Art von Listener gibt es eine eigene Registrierungsmethode.
- (K) Die Klasse Canvas bietet eine unbegrenzte Zeichenfläche in einem Fenster.

V2 Vereinfachung mittels Lambda-Ausdrücken

★ ☆ ☆

Gegeben sei nachstehender Java-Code. In diesem wird aus einer Liste von Zahlen der Durchschnittswert aller *positiven geraden* Zahlen berechnet, in einer Variablen gespeichert und zurückgegeben.

```

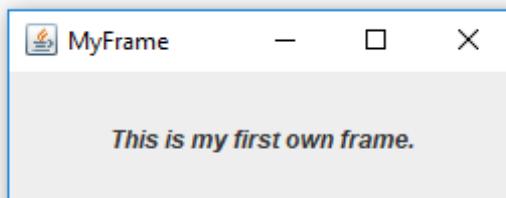
1 double averageOfEvenNumbers(int[] numbers) {
2     int sum = 0, count = 0;
3     for (int i = 0; i < numbers.length; i++) {
4         if (numbers[i] > 0 && numbers[i] % 2 == 0) {
5             sum += numbers[i];
6             count++;
7         }
8     }
9     return ((double)sum) / count;
10 }
```

1. Nutzen Sie Streams und Lambda-Ausdrücke um den Code zu verkürzen.
2. Beide Implementierungen – die “normale“ und die Stream-Variante – weisen eine konzeptionelle Unsauberkeit auf. Bei beiden kann es zum Auftreten einer Exception kommen.
 - (a) An welcher Stelle kann es bei der “normalen“ Implementierung zu einer Exception kommen? Um welche handelt es sich?
 - (b) An welcher Stelle kann es bei der Implementierung mittels Streams zu einer Exception kommen? Um welche handelt es sich?
3. Modifizieren Sie den Code entsprechend, dass die Exceptions aus Teilaufgabe 2 gar nicht mehr auftreten können, sondern die Methode für alle übergebenen Arrays fehlerfrei durchläuft.

V3 Mein eigenes kleines Fenster

★ ☆ ☆

Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



V4 Innere Klassen und Scope



Betrachten Sie den untenstehenden Java-Code, welcher eine innere Klasse `InnerClass` enthält. In dieser wiederum ist eine Methode definiert, die einen Lambda-Ausdruck enthält, dessen Operation mit der Methode `accept` des Interfaces `Consumer` aufgerufen wird.

```

1 import java.util.function.Consumer;
2 public class LambdaScope {
3     public int x = 0;
4
5     class InnerClass {
6         public int x = 11;
7
8         void methodInInnerClass(int x) {
9             int z = 55;
10
11            Consumer<Integer> myConsumer = (y) ->
12            {
13                System.out.println("x = " + x);
14                System.out.println("y = " + y);
15                System.out.println("this.x = " + this.x);
16                System.out.println("LambdaScope.this.x = " +
17                    LambdaScope.this.x);
18                System.out.println("z = " + z);
19            };
20            myConsumer.accept(x);
21        }
22
23    public static void main(String[] args) {
24        LambdaScope scope = new LambdaScope();
25        LambdaScope.InnerClass f1 = scope.new InnerClass();
26        f1.methodInInnerClass(123);
27    }
28 }
```

Welche Ausgabe wird bei der Ausführung des Codes auf der Konsole ausgegeben? Überlegen Sie sich die Antworten ohne die Nutzung eines Compilers!

V5 Zeilen nummerieren



Implementieren Sie die Methode `void insertRowNumbers(String path)`, welche den Pfad einer Text-Datei als String übergeben bekommt. Die Methode soll den Text der Datei Zeile für Zeile einlesen. Beginnend ab der ersten Zeile soll dann in jeder zweiten Zeile nun die Zeilennummer eingefügt werden. War die alte Text-Datei folgendermaßen aufgebaut `"Row1 \n Row2"` so soll die neue Text-Datei so aussehen: `"1 \n Row1 \n 2 \n Row2"`. Dabei steht `"\n"` für einen Zeilenumbruch.

V6 Bäckerei gesucht



Sie planen eine Party und benötigen eine Menge Brötchen dafür. Sie suchen nun den Bäcker in Ihrer Umgebung, der Ihnen die günstigsten Brötchen verkaufen kann. Dafür gibt es Objekte des Typs `Bakery`, welche zum einen zwei `public double`-Attribute `distance` (gibt die Distanz zu Ihnen in km an) und `price` (gibt den Preis pro Brötchen an) besitzt. Außerdem gibt es Objekte des Typs `BakeryOffer`, welche ebenfalls zwei `public`-Attribute `bakery` vom Typ `Bakery` und `totalPrice` vom Typ `double` besitzt. Der Konstruktor der Klasse sieht folgendermaßen aus:

```
public BakeryOffer(Bakery b, double tp){  
    this.bakery = b;  
    this.totalPrice = tp; }
```

Implementieren Sie nun eine `public`-Methode vom Rückgabetyp `List<BakeryOffer>` mit dem Methodenkopf:

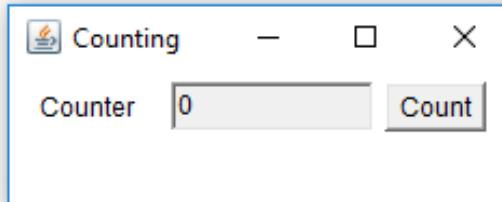
```
sortedBakeryOffers(Collection<Bakery> bakeries, double maxDistance)
```

Die Methode erhält eine Sammlung von Bäckereien und die maximale Distanz zu einer solchen. Zurückgeliefert werden soll eine nach Gesamtpreis aufsteigend sortierte Liste mit Angeboten des Typs `BakeryOffer`. Bäckereien, die weiter als die übergebene maximale Distanz entfernt sind, sollen von der Betrachtung ausgeschlossen werden.

V7 Zähler



Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



Jedes Mal wenn der `Count`-Button gedrückt wird, soll sich der Wert im Feld um 1 erhöhen.

Hinweis: Ihr Programm besteht aus drei Komponenten:

1. `java.awt.Label "Counter"`
2. non-editable¹ `java.awt.TextField`
3. `java.awt.Button "Count"`

¹http://programmedlessons.org/java5/Notes/chap60/ch60_13.html

H Elfte Hausübung

Campus-Management

Gesamt 7 Punkte

In dieser Hausübung befassen wir uns mit einem kleinen, selbstgeschriebenen Campus Management System, welches Sie in Abbildung 1 sehen.

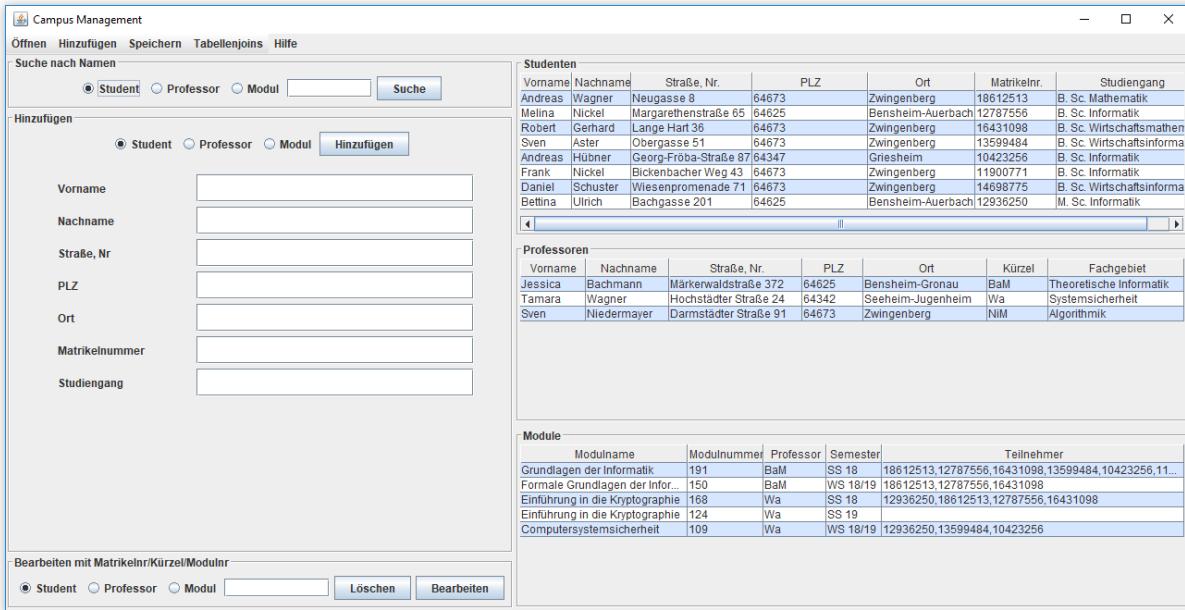


Abbildung 1: Screenshot aus dem Hauptmenü unseres Campus Management Systems.

Wie Sie leicht sehen können, verwaltet unser System drei Objektarten - Studenten, Professoren und Module. Jede dieser Gruppen hat dabei verschiedene Attribute, welches sie klassifiziert. Die Hauptfunktionen unseres Systems liegen darin, dass Einträge dieser Gruppen hinzugefügt, bearbeitet oder gelöscht werden können. Gespeichert werden die Daten dann in Dateien des Typs .txt. Um diese Dateien zu öffnen oder zu speichern, stehen in der oberen Taskleiste Drop-Down Menüs bereit.

H1 Entdecken des Campus Management Systems 0 Punkte

Verschaffen Sie sich zunächst einen Überblick über das System. Die graphische Oberfläche ist bereits fertig gestaltet, Sie müssen zunächst also nur die Funktionen selbst implementieren.

Werfen Sie also neben dem Blick auf die GUI auch einen tieferen Blick in die Codebasis! Die angegebenen relevanten Foliensätze sollten Ihnen dabei enorm helfen.

H2 Suche**2 Punkte**

Ergänzen Sie in der Klasse `Manager` die drei Methoden:

- `public ArrayList<Student> searchStudentName(String name)`
- `public ArrayList<Professor> searchProfName(String name)`
- `public ArrayList<Module> searchModuleName(String name)`

In der Klasse gibt es drei `ArrayLists` namens `students`, `profs` und `modules`. Diese Listen sollen dahingehend durchsucht werden, ob der übergebene String `name` Teilstring der jeweiligen Namen ist. Bei den Studenten und Professoren durchsuchen Sie den Vor- und Nachnamen nach dem String, bei den Modulen den Modulnamen. Zurückgegeben wird eine Liste von Studenten/Professoren/Modulen, bei denen im Vor-, Nach- oder Modulnamen der String `name` gefunden wurde.

Wurde in einer der drei Methoden nichts gefunden, so ist über die Methode `showMessageDialog` der Klasse `JOptionPane` eine Meldung auszugeben. Für Studenten könnte diese beispielsweise folgendermaßen lauten:

"Es gibt keinen Studenten mit Teil - Namen: xyz"

H3 Anzeigen**1 Punkt**

Ergänzen Sie in der Klasse `Manager` die drei Methoden:

- `public void showStudents(ArrayList<Student> list)`
- `public void showProfs(ArrayList<Professor> list)`
- `public void showModules(ArrayList<Module> list)`

Die drei Methoden sollen die jeweils übergebene Liste in den MainFrame schreiben. Dabei können Sie die Liste des MainFrames über `mainFrame.studentModel` beziehungsweise `mainFrame.profModel` oder `mainFrame.moduleModel` ansprechen. Sollten gerade schon Objekte im MainFrame angezeigt werden, so löschen Sie diese zunächst und zeigen Sie danach die neuen Objekte der übergebenen Liste an.

Ist die Länge der übergebenen Liste ungleich der Länge der Klassenattribute `students`, `profs` und `modules` so setzen Sie das Attribut `filteredStudents` bzw. `filteredProfs` oder `filteredModules` auf `list`, andernfalls auf `null`.

Nutzen Sie am Ende `mainFrame.resizeColumnWidth(mainFrame.studentTable)` bzw. `mainFrame.profTable` oder `mainFrame.moduleTable`, um die Tabellenbreite anzupassen.

H4 SearchHandler**2 Punkte**

Ergänzen Sie die Methode `public void actionPerformed(ActionEvent e)` in der Klasse `SearchHandler`. Greifen Sie zunächst auf den Text zu, der in das Suchfeld namens `searchfield` im MainFrame eingetippt wurde. Ist das Suchfeld leer, so geben Sie wieder eine Meldung über `ShowMessageDialog` aus. Andernfalls machen Sie eine Fallunterscheidung, welcher der drei Buttons Student/Professor/Modul ausgewählt wurde. Nach dem Klick auf den *Suche* Button sollen nur noch gefilterte Ergebnisse angezeigt werden, welche im Namen den gesuchten String enthalten.

Hinweis:

Setzen Sie an dieser Stelle auch die Attribute `manager.filterStudents`, `manager.filterProfs` und `filterModuls` geeignet.

H5 Neue Suchfunktion**2 Punkte**

Zum Abschluss eine kreativere Aufgabe. Erweitern Sie die Suchfunktion des Campus Management Systems, sodass auch nach anderen Attributen als dem Namen gesucht werden kann. Überlegen Sie sich selbst eine Darstellungsmöglichkeit (z.B. Drop-Down-Menüs o.Ä.), welche Ihnen hier passend erscheint.

Beachten Sie außerdem, dass Studenten, Professoren und Module alle andere Attribute besitzen. Der Nutzer muss also je nachdem welche Gruppe ausgewählt ist, aus den passenden Attributen zur Suche wählen können.

Um die beiden Punkte dieser Aufgabe zu erhalten, muss folgendes umgesetzt werden:

1. Die erweiterte Suchfunktion muss in der GUI auswählbar sein und die passenden Attribute für die jeweiligen Suchen anzeigen.
2. Die Funktionalität muss korrekt implementiert sein.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrlig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.1

Übungsblatt 12

Themen: Wrap-Up II

Relevante Folien: alle bis dahin

Abgabe der Hausübung: 08.02.2019 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Schleifenerkennung in verzeigerten Strukturen

Für diese Aufgabe betrachten wir folgende Klasse für Listenelemente, die Sie auch schon in der Vorlesung und auf Übungsblatt 9 kennengelernt haben.

```
public class ListItem<T>{
    public T key;
    public ListItem<T> next;
}
```

In einer solchen Liste kann es theoretisch vorkommen, dass eine Schleife vorkommt. Damit ist gemeint, dass ein Element der Liste auf ein früheres zurückverweist. Somit kommt man beim linearen Durchlaufen der Liste in eine Schleife. Ein Beispiel dafür finden Sie in Abbildung 1.

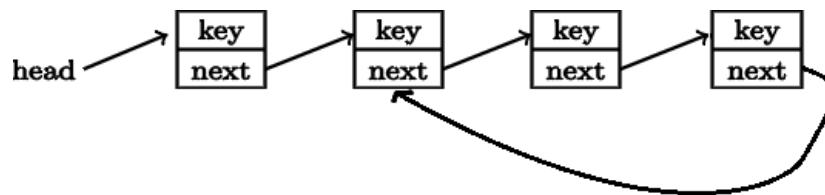


Abbildung 1: Eigene LinkedList-Klasse aus der Vorlesung mit Schleife.

In dieser Aufgabe soll es darum gehen verschiedene Ansätze kennenzulernen und zu vergleichen, mit denen solche Schleifen in linearen Listen entdeckt werden können.

Setzen Sie alle nachfolgend vorgestellten Ansätze in Java um! Diskutieren Sie anschließend die Vor- und Nachteile jedes Ansatzes.

Funktioniert überhaupt jeder der Ansätze? Wo kann es zu Problemen kommen?

V1.1 Jedes Element merken

Die erste Möglichkeit besteht darin jedes einzelne Element, welches bereits durchlaufen wurde, abzuspeichern. Bei jedem neuen Schritt wird dann geschaut, ob das Element bereits abgespeichert wurde. Zum Abspeichern kann in Java beispielsweise ein `HashSet` verwendet werden.

V1.2 Doppelte Verkettung nutzen

Sie können die Klasse für Listenelemente dahingehend erweitern, dass es nun zusätzlich einen Zeiger auf das vorherige Element gibt. Beim Durchlauf der Liste muss nun jedes Element das zuvor durchlaufende Element referenzieren, ansonsten liegt eine Schleife vor.

V1.3 Vergleich mit Startelement

Der Zeiger auf das nächste Element wird mit dem Startelement der Liste verglichen. Sind diese beiden Elemente gleich, so liegt eine Schleife vor.

V1.4 Jedes Element markieren

Die Klasse für Listenelemente soll um ein Attribut `public boolean visited` erweitert werden. Dieses ist initial auf `false` gesetzt und wird beim Durchlaufen auf die bereits besuchten Elemente auf `true` gesetzt. Kommen wir bei einem Element vorbei, welches bereits auf `true` gesetzt ist, muss eine Schleife vorliegen.

V1.5 Liste sortieren

Zunächst wird die Liste mittels Comparator aufsteigend sortiert. Anschließend wird beim Durchlauf jedes Element mit dem nächsten verglichen. Wenn der Zeiger auf ein kleineres Element verweist, muss eine Schleife vorliegen.

V1.6 Hase-Igel-Algorithmus

Der Algorithmus besteht aus dem gleichzeitigen Durchlauf der Liste mit unterschiedlichen Schrittweiten. Dabei werden zwei Zeiger auf Listenelemente benutzt, von denen der eine (Igel) bei jeder Iteration auf das nächste Element verschoben wird, während der andere (Hase) bei derselben Iteration auf das übernächste Element verschoben wird. Wenn die beiden Zeiger sich begegnen, also dasselbe Element referenzieren, hat die Liste eine Schleife. Wenn einer der beiden Zeiger das Ende der Liste erreicht, hat sie keine Schleife.

Implementieren Sie die Methode `boolean tortoiseAndHare(ListItem<T> head)`, welche den Kopf einer Liste übergeben bekommt und `true` genau dann zurückliefert, wenn die Liste einen Zyklus enthält.

V2 Objektorientierte Modellierung

In dieser Aufgabe geht es um das Athene-Bistro im Robert-Piloty-Gebäude. Das Bistro bietet verschiedene Speisen und Getränke an, welche Sie in der folgenden Aufgabe sinnvoll gemäß des objektorientierten Programmierkonzeptes modellieren sollen. Erstellen Sie eine

Typhierarchie (vergleichen Sie dazu nochmal die Aufgaben V3, V5 und V6 von Blatt 8) welche mindestens die folgenden konkreten Elemente umfasst:

- die Getränketypen Softdrink (hier gibt es Cola und Fanta) und Bier
- die Snacks Schokoriegel (hier gibt es Snickers, Mars und Kinderschokolade), Chips und Eis
- die Speisen Bockwurst, Sandwich und Schnitzelbrötchen

Beachten Sie zur Modellierung folgende Hinweise:

- Auf Methoden und Attribute dürfen Sie in dieser Aufgabe verzichten.
- Fügen Sie sinnvolle und möglichst passend benannte Obertypen ein.
- Markieren Sie abstrakte Klassen und Interfaces eindeutig!
- Es gibt bei dieser Aufgabe keine „richtige“ Lösung. Begründen Sie, wieso Sie sich für genau diese Modellierung entschieden haben und was man eventuell anders hätte gestalten können.

V3 Absteigend sortiert?

Ergänzen Sie die folgende Methode und ihre Paramterliste:

```
1 /**
2  * Checks whether the given list is sorted in descending
3  * order.
4  * @param alox the list which has to be checked.
5  * @return true, if the list is sorted in descending order.
6 */
7 public boolean isSortedDescending(final List<T> alox) {
8
9 }
```

Ihre Methode muss mit generischen Listen umgehen können. Wenn die übergebene Liste leer ist, so ist `true` zurückzuliefern. Sie dürfen weiterhin davon ausgehen, dass die Liste nur Elemente enthält, welche nicht `null` sind.

V4 Map in Java

In Racket haben Sie zu Beginn des Semesters eine Funktion höherer Ordnung namens `map` kennengelernt. Außerdem haben Sie in der Vorlesung zunächst eine eigene Implementierung dieser Funktion gesehen. Zur Erinnerung:

```
;; Type: (X -> Y) (list of X) -> (list of Y)
(define (my-map fct list)
  (cond
    [(empty? list) empty]
    [else (cons (fct (first list))
                (my-map fct (rest list)))]))
```

Auch in Java können wir mittels Streams ganz einfach auf die `map` Methode zugreifen. In dieser Aufgabe sollen Sie jedoch die Funktionalität selbst implementieren. Ergänzen Sie dazu den folgenden Code:

```
/** 
 * Applies a method to every element of a list.
 * @param fct function to apply
 * @param list list to iterate over
 * @return [fct(list[0]), fct(list[1]), ..., fct(list[n-1])]
 */
public <T,R> List<R> map(Function<T,R> fct, List<T> list) {
}
```

Die Funktion `fct` realisieren Sie hierbei später über einen Lambda-Ausdruck.

Folgender Aufruf in Racket...

```
(define number-list (list 3 6 9))
(my-map (lambda (x) (* x 10)) number-list)
```

... wird dann später in Java folgendermaßen umgesetzt:

```
Function<Integer, Integer> multiply10 = new Function<>() {
    public Integer apply(Integer i) { return i*10; }
};

map(i -> i.multiply10(),
     Arrays.asList(new Integer[] {3, 6, 9}));
```

V5 Generics

Gegeben seien kurze Codeauszüge. Geben sie bei jedem der Auszüge (ohne Hilfe des Compilers!) an, ob der Code kompiliert. Wenn nicht, dann begründen Sie kurz wieso.

V5.1

```
public final class bar {
    public static <T> T oracle(T x, T y) {
        return x > y ? x : y;
    }
}
```

V5.2

```
public static void print(List<? extends X> alox) {
    for (X x : alox)
        System.out.print(x + " ");
}
```

V5.3

```
public class Y<T> {

    private static T y = null;

    public static T foo() {
        if (y == null)
            y = new Y<T>();

        return y;
    }
}
```

V5.4

```
class A {}
class B extends A {}
class C extends A {}
class D<T> {}

D<B> db = new D<>();
D<A> da = db;
```

H Zwölfte Hausübung

Gesamt 12 Punkte

Graphen mit Adjazenzmatrizen

In dieser Hausübung beschäftigen wir uns erneut mit dem Thema Graphen, welche Ihnen bereits aus der Hausübung des ersten Wrap-Up Blatts bekannt sind. Als Graph bezeichnet man in der Informatik eine Datenstruktur, die eine Menge von Objekten, sowie eine Menge von Verbindungen zwischen diesen, speichern kann. Die Objekte eines Graphen werden Knoten genannt, die paarweisen Verbindungen zwischen den Knoten nennt man Kanten.

Um abzuspeichern, welche Knoten miteinander verbunden sind, wollen wir eine sogenannte Adjazenzmatrix (oder auch Nachbarschaftsmatrix) verwenden. Die Adjazenzmatrix eines Graphen ist eine Matrix, die speichert, welche Knoten des Graphen durch eine Kante verbunden sind. Sie besitzt für jeden Knoten eine Zeile und eine Spalte, woraus sich für n Knoten eine Matrix der Größe $n \times n$ ergibt. Ein Eintrag in der i -ten Zeile und j -ten Spalte gibt hierbei an, ob eine Kante vom i -ten zum j -ten Knoten führt. Steht an dieser Stelle *NaN* (Not a Number), ist keine Kante vorhanden – eine Zahl gibt an, dass eine Kante existiert und repräsentiert gleichzeitig das sogenannte Kantengewicht. In den folgenden Abbildungen sehen Sie einen Beispielgraphen, sowie seine zugehörige Adjazenzmatrix.

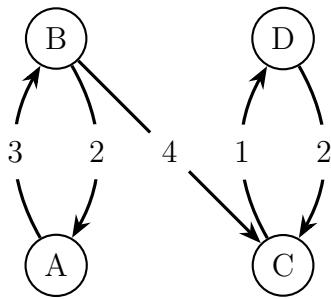


Abbildung 2: Ein Beispielgraph mit vier Knoten.

	A	B	C	D
A	NaN	3	NaN	NaN
B	2	NaN	4	NaN
C	NaN	NaN	NaN	1
D	NaN	NaN	2	NaN

Abbildung 3: Adjazenzmatrix

Ihr Ziel in dieser Aufgabe ist es, eine Graphenklasse in Java von Grund auf zu implementieren. Um die Klasse möglichst vielfältig einsetzen zu können, werden Sie die Klasse sinnvoll generisch parametrisieren müssen. Wir wollen uns nicht auf einen konkreten Typ für die Knoten eines Graphen festlegen, die Klasse soll mit jedem beliebigen Typ umgehen können. Sie können dabei frei entscheiden, wie Sie die Knoten in der Klasse abspeichern. Die Kantengewichte eines Graphen wollen wir ebenfalls nicht auf einen konkreten Typ festlegen, jeder Typ, der das **Comparable**-Interface implementiert, ist zulässig. Zusätzlich soll es in unserer Graphen-Klasse möglich sein, dass ein Graph bei Bedarf mehrere Adjazenzmatrizen und somit auch mehrere Kantenmengen besitzt. Folgend ein kurzes Beispiel dafür: Wir betrachten einen Graph G , der zur Berechnung von Navigationsrouten verwendet werden soll. G besitzt zwei Adjazenzmatrizen A_{time} und A_{distance} und dadurch ist es uns möglich die kürzeste Strecke, gemessen in Zeit über die Matrix A_{time} , oder gemessen am Weg über die Matrix A_{distance} zu berechnen. Die Adjazenzmatrizen eines Graphen werden über einen eindeutigen Bezeichner von Typ **String** identifiziert, in unserem Beispiel "**time**" und "**distance**". Ein Objekt der Graphenklasse darf nicht mehrere Adjazenzmatrizen mit dem gleichen Bezeichner besitzen. Überlegen sie sich, wie Sie mehrere Matrizen speichern können und wie Sie diese Matrizen über einen Bezeichner von Typ **String** identifizieren können.

H1 Modellierung**2 Punkte**

Erstellen Sie nun zunächst eine Klasse `Graph` im Package `Main`. Modifizieren Sie die Klasse dahingehend, dass die aus dem obigen Einleitungstext genannten Anforderungen erfüllt werden können. Zusätzlich fügen Sie der Klasse einen Konstruktor hinzu. Dieser bekommt als einzigen Parameter das Kantengewicht übergeben, dass für eine nicht-existierende Kante steht. In dem Beispielgraph aus Abbildung 2 wäre dies `NaN`. Nach dem Aufruf des Konstruktors ist die Anzahl an Kanten und Knoten gleich null.

H2 Methode addNode**2 Punkte**

Um unseren Graphen befüllen zu können, schreiben Sie nun eine Methode `public void addNode`. Die Methode bekommt einen Knoten übergeben und fügt diesen dem Graphen hinzu. Passen Sie alle Adjazenzmatrizen diesbezüglich an. Alle Kantengewichte werden mit dem im Konstruktor übergebenen Parameter initialisiert. Dem ersten Knoten der dem Graphen hinzugefügt wurde, wird Reihe 0 und Spalte 0 in der Adjazenzmatrix zugewiesen, dem zweiten Knoten wird Reihe 1 und Spalte 1 zugewiesen usw.

H3 Methode setWeight**2 Punkte**

Nachdem bereits das Einfügen neuer Knoten in den Graphen implementiert wurde, kümmern wir uns nun um die Kantensetzung zwischen diesen. Dazu schreiben Sie eine Methode `public void setWeight`. Die Methode bekommt als ersten Parameter den Bezeichner einer Adjazenzmatrix des Graphen als `String` übergeben. Als zweiten Parameter bekommt sie die Reihe der Matrix und als dritten Parameter die Spalte der Matrix jeweils als `int` übergeben. Der vierte Parameter ist das Kantengewicht, dass der Kante in der übergebenen Reihe und Spalte zugewiesen werden soll. Besitzt der Graph noch keine Adjazenzmatrix zum übergebenen Bezeichner, so ist eine neue Adjazenzmatrix für diesen Bezeichner einzurichten um dann das Kantengewicht zu setzen. Alle anderen Kantengewichte, der neuen Matrix, werden mit dem im Konstruktor übergebenen Parameter initialisiert. Sie dürfen davon ausgehen, dass die übergebene Reihe und Spalte kleiner ist als die aktuelle Anzahl an Knoten und müssen diesen Fall nicht gesondert behandeln.

H4 Methode getWeight und getNumberOfNodes**1 Punkt**

Implementieren Sie eine `public`-Methode `getWeight`. Die Methode bekommt als ersten Parameter den Bezeichner einer Adjazenzmatrix des Graphen als `String` übergeben. Als zweiten Parameter bekommt sie die Reihe der Matrix als `int` übergeben. Als dritten Parameter die Spalte der Matrix als `int` übergeben. Zurückgegeben wird das Kantengewicht, dass sich an der übergebenen Reihe und Spalte in der Matrix mit dem übergebenen Bezeichner befindet. Sie dürfen davon ausgehen, dass eine Matrix mit dem übergebenen Bezeichner im Graphen existiert und dass die übergebene Reihe und Spalte kleiner ist als die aktuelle Anzahl an Knoten. Sie müssen diesen Fällen nicht gesondert behandeln.

Implementieren Sie eine Methode `public int getNumberOfNodes`. Die Methode hat keine Parameter und soll lediglich die aktuelle Anzahl an Knoten im Graph zurückgeben.

H5 Methode getAllPaths**3 Punkte**

Implementieren Sie nun eine Methode `public String[] getAllPaths`. Die Methode bekommt als ersten Parameter den Bezeichner einer Adjazenzmatrix des Graphen als `String` übergeben. Als zweiten Parameter bekommt sie den Index eines Knotens im Graphen als `int` übergeben. Der Index meint hier, welche Reihe und Spalte in der Adjazenzmatrix dem Knoten zugeordnet wird. Als dritten und letzten Parameter bekommt die Methode einen `boolean` übergeben, später dazu mehr. Die Methode soll ausgehend von dem übergebenen Startknoten alle möglichen Pfade finden, in denen kein Knoten mehrmals vorkommt. Dabei ist zu beachten, dass keiner der gefundenen Pfade ein Teilstück eines anderen gefundenen Pfades ist. Zurückgegeben werden soll ein Array von `Strings`. Jeder String dieses Arrays steht dabei für einen Pfad. Die Reihenfolge in der die Pfade im Array vorkommen kann beliebig sein. Sie dürfen davon ausgehen, dass eine Matrix mit dem übergebenen Bezeichner im Graphen existiert und müssen diesen Fall nicht gesondert behandeln.

Folgend ein Beispiel: Wir gehen davon aus, dass wir ein Graphen-Objekt mit den Knoten und Kanten aus Abbildung 2 eingerichtet haben, die dazugehörige Adjazenzmatrix wird mit dem `String "example"` identifiziert. Nun betrachten wir einen Aufruf der Methode `getAllPaths` dieses Objekts. Wir übergeben `"example"` als Bezeichner einer Adjazenzmatrix und `1` als den Index eines Knotens (Knoten B aus Abbildung 2), den dritten Parameter betrachten wir zunächst nicht. Der Methodenaufruf liefert den Pfad von B nach A und den Pfad von B nach C nach D zurück.

Die Pfade als `String` sollen nach folgendem Schema formatiert werden: `"N1 -> N2 -> ... -> Nk"`. Die Kanten eines Pfades werden durch den String `" -> "` (Leerzeichen, Bindestrich, geschlossene spitze Klammer, Leerzeichen) angegeben. Die Knoten werden entweder durch ihren zugehörigen Index in der Adjazenzmatrix oder durch die `toString()`-Methode der Knoten angegeben. Ist der dritte Parameter der Methode `true`, so werden die Indizes der Knoten im Pfad-String angegeben, andernfalls wird die `toString()`-Methode der Knoten im Pfad-String angegeben.

Gehen wir nun davon aus, dass in unserem obigen Beispiel die Knoten des Graphen von Typ `String` sind, dann sieht unser zurückgegebenes Array folgendermaßen aus:

Wenn der dritter Parameter == `false` (Knoten dargestellt mittels `toString()`-Methode):

`{"B -> A", "B -> C -> D"}.`

Wenn der dritter Parameter == `true` (Knoten als Index):

`{"1 -> 0", "1 -> 2 -> 3"}.`

Der Pfad `"1 -> 2"` oder eben `"B -> C"`, darf im zurückgegebenen Array nicht vorhanden sein, da er ein Teilstück von `"1 -> 2 -> 3"` oder eben `"B -> C -> D"` ist.

H6 Testen**2 Punkte**

Abschließend wollen wir unsere implementierte Klasse testen. Dazu erstellen Sie eine neue Klasse `GraphTests` im Package `Main`. Implementieren Sie nun jeweils mindestens 3 Tests für die Methoden `addNode`, `setWeight` sowie `getAllPaths`.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.5

Übungsblatt 13

Themen: Wrap-Up III

Relevante Folien: alle bis dahin

Abgabe der Hausübung: 15.02.2019 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Bildverarbeitung

In dieser Aufgabe wollen wir uns mit dem Filtern von Bildern beschäftigen. Dies ist in der digitalen Bildverarbeitung eine häufig verwendete Technik um beispielsweise Bilder zu schärfen oder zu blurren¹. Zur Bearbeitung dieser Aufgabe finden Sie eine Vorlage namens UE13_V1 in moodle.

Portable Graymaps

Als Datenformat für die verwendeten Bildern benutzen wir hier Portable Graymaps. Das PGM-Format dient zur Speicherung von Rastergrafiken und kann Bilder in Graustufen darstellen. Eine PGM Datei ist dabei immer folgendermaßen aufgebaut:

```
Magischer Wert (Steht fuer das Format der Bilddaten)
# Kommentare
Breite Hoehe
Maximalwert der Helligkeit

Bilddaten in Form von Pixel
```

Als Beispiel sehen Sie sich Abbildung 1 an. Dort ist das Kürzel FOP durch folgende PGM-Datei kodiert:

```
P2
# Das Kuerzel FOP kodiert als PGM
18 7
15
```

¹Blurring = Weichzeichnen

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	3	3	3	0	0	7	7	7	7	0	0	0	15	15	15	15	0
0	3	0	0	0	0	0	7	0	0	7	0	0	0	15	0	0	15	0
0	3	3	3	0	0	0	7	0	0	7	0	0	0	15	15	15	15	0
0	3	0	0	0	0	0	7	0	0	7	0	0	0	15	0	0	0	0
0	3	0	0	0	0	0	7	7	7	7	0	0	0	15	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

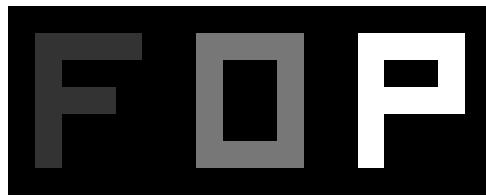


Abbildung 1: FOP als PGM-Datei

PGM-Dateien können mit einem Bildeditor wie beispielsweise GIMP visualisiert werden.

Um das Einlesen der Bilder brauchen Sie sich in dieser Übung nicht zu kümmern, das übernimmt die von uns vordefinierte Klasse `Image` für Sie. Diese enthält folgende Attribute:

- `String magicNumber`: Die magische Zahl, welche das Format der Bilddatei angibt (kann von Ihnen ignoriert werden)
- `int width`: Die Breite des Bildes (Anzahl an Pixeln)
- `int height`: Die Höhe des Bildes (Anzahl an Pixeln)
- `float max`: Der maximale Helligkeitswert des Bildes
- `float[][] data`: Die Bilddaten als Pixel in einem 2-dimensionalen Array gespeichert (erste Dimension Höhe, zweite Breite)

Außerdem liefert Sie folgende Methoden:

- Die Getter `String getMagicNumber()`, `float getData()`, `float getMax()`, `int getWidth()` und `int getHeight()`, welche die jeweiligen Attribute zurückliefern
- Die Methode `float getPixel(int height, int width)`, welche den Pixelwert an der gegebenen Stelle zurückgibt
- Die Methode `float setPixel(float value, int height, int width)`, welche den Pixelwert an der gegebenen Stelle überschreibt
- Der Konstruktor `Image(Path p)`, welche den Pfad zur Bilddatei entgegennimmt und einliest
- Die Methode `save(Path p)`, welche das Bild am angegebenen Pfad abspeichert

Alles andere in der Klasse ist für Sie nicht von Bedeutung, ändern Sie in dieser Klasse auch in keinem Fall etwas ab! Wir liefern Ihnen in dieser Übung mehrere Beispielbilder mit, mit denen auch unsere Tests laufen.

V1.1 Lineares Filtern

Beim linearen Filter wird die sogenannte diskrete Faltung verwendet. Dies funktioniert folgendermaßen:

Das Eingabebild wird durch ein zweidimensionales Array repräsentiert, welches wir als Bildmatrix B bezeichnen. Für das Beispiel in Abbildung 1 sieht der Anfang der Bildmatrix folgendermaßen aus:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 3 & 3 & 3 & \dots \\ 0 & 3 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Betrachten Sie zur besseren Verständlichkeit Abbildung 2, in der die Berechnung eines Pixels exemplarisch vorgemacht ist.

13	7	4	13	2	13	5
2	6	5	6	15	3	3
12	9	14	15	5	12	13
12	8	12	10	6	7	11
7	11	15	7	7	10	2
4	1	9	1	12	6	9
12	2	5	10	10	10	7

0	-1	0
-1	5	-1
0	-1	0

13	7	4	13	2	13	5
2	6	5	6	15	3	3
12	9	29	15	5	12	13
12	8	12	10	6	7	11
7	11	15	7	7	10	2
4	1	9	1	12	6	9
12	2	5	10	10	10	7

Abbildung 2

Auf die Bildmatrix B wird eine Kernelmatrix K angewendet. Dabei wird jeder Pixel des Ursprungsbildes durch eine Linearkombination seiner Nachbarschaft und sich selbst ersetzt. Die Einträge der Kernelmatrix K stellen dabei die Gewichte dieser Linearkombination dar. Bezeichnen wir A als Ausgangsbild und mit $A(i, j)$ den Pixel in der i -ten Zeile und in der j -ten Spalte, dann ist die diskrete Faltung definiert als: $A := B * K$. Die neuen Pixelwerte des Ausgangsbildes berechnen sich durch:

$$A(i, j) = \sum_{x=1}^n \sum_{y=1}^n B(i + x - a_1, j + y - a_2) \cdot K(x, y)$$

In der Formel oben steht dabei n für die Filtergröße, a_1 für die x-Koordinate und a_2 für die y-Koordinate des Kernelmittelpunktes.

Für Pixelwerte am Rand ist der Kernel eventuell zu groß und greift ins Leere. Dafür wird eine sogenannte Randbehandlungsstrategie benötigt. Die Werte außerhalb des Bildrandes werden dann einfach dem Wert des örtlich nächsten Nachbarns auf dem Bildrand gleichgesetzt. Betrachten Sie Abbildung 3 für diesen Spezialfall.

8 8 11	
8 8 11	0 12 9 0 10
0 0 2 15 2 12 14 3	
6 0 9 15 5 8 0	
14 2 11 6 0 12 4	
5 7 10 5 10 12 13	
1 7 12 3 8 13 13	
15 1 8 7 6 8 8	

0 -1 0	
-1 5 -1	
0 -1 0	

13 11 0 12 9 0 10	
0 2 15 2 12 14 3	
6 0 9 15 5 8 0	
14 2 11 6 0 12 4	
5 7 10 5 10 12 13	
1 7 12 3 8 13 13	
15 1 8 7 6 8 8	

Abbildung 3

Hinweis: Sollte durch das Filtern keine natürliche Zahl herauskommen, wird diese auf die nächst kleinere ganze Zahl abgerundet. Weiterhin müssen Werte kleiner als 0 auf 0 und Werte größer als das Maximum auf das Maximum gesetzt werden.

In Abbildung 4 sehen Sie ein Beispielbild, welches mittels diskreter Faltung geschärft und weichgezeichnet wurde.



Ausgangsbild

Schärfung

Weichzeichner

Abbildung 4

Beispiele für solche Kernel sind:

$$\text{Schaerfen: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{Weichzeichnen: } \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

V1.2 Implementieren des Linearen Filters

Ergänzen Sie die Methode `linearFilter(Image image, float[][][] kernel)` in der Klasse `Filter`, die das lineare Filtern mithilfe der diskreten Faltung implementiert. Beachten Sie hierbei auch die Behandlung des Bildrandes wie oben beschrieben.

V1.3 Nichtlineares Filtern

Das nichtlineare Filtern stellt eine Besonderheit des Filterns dar. Anstelle einer Linear-kombination betrachten wir hier nur die Nachbarschaft des aktuellen Pixels und verwenden hier einen sogenannten Rangordnungsfilter. In dieser Hausübung beschäftigen Sie sich mit dem Medianfilter. Ein $n \times n$ - Medianfilter sucht sich alle Pixel in einer Umgebung der Größe $n \times n$ und bringt diese in eine Rangordnung. Anschließend wird der Median dieser Rangordnung gebildet und der Pixel durch diesen Medianwert ersetzt. Als Beispiel betrachten Sie Abbildung 5 für einen 3×3 Medianfilter.

3	14	13	7	5	1	15
15	12	7	13	11	8	11
9	10	5	14	12	4	12
14	0	2	2	4	1	13
4	4	8	3	3	9	7
2	9	1	7	0	2	4
2	2	15	12	2	7	0

12	7	13	10	5	14	0	2	2
↓ sortiert ↓								
0	2	2	5	7	10	12	13	14
↑								
Median								

3	14	13	7	5	1	15
15	12	7	13	11	8	11
9	10	7	14	12	4	12
14	0	2	2	4	1	13
4	4	8	3	3	9	7
2	9	1	7	0	2	4
2	2	15	12	2	7	0

Abbildung 5

Hier wird die identische Randbehandlungsstrategie wie beim Linearen Filtern verwendet. Sie ersetzen also Pixelwerte außerhalb des Bildrandes mit den örtlich lokalen Nachbarn des Bildrandes. In Abbildung 6 sehen Sie ein Bild welches mittels eines 5×5 und eines 9×9 Medianfilters weichgezeichnet wurde.



Ausgangsbild

 5×5 Medianfilter 9×9 Medianfilter

Abbildung 6

V1.4 Lokale Nachbarschaft

Zur einfacheren Implementierung des Medianfilters steht Ihnen die Klasse `LocalNeighbours` zur Verfügung. Diese dient der Speicherung der aktuellen Nachbarschaftspixel. Folgende Methoden stehen Ihnen hier zur Verfügung:

- Die Methode `void addPixel(float Pixel)` fügt der lokalen Nachbarschaft einen neuen Pixelwert hinzu.
- Die Methode `int getSize()` gibt die Anzahl der abgespeicherten Pixel in der lokalen Nachbarschaft zurück.
- Die Methode `Float[] getPixels()` gibt ein Array mit den gespeicherten Pixelwerten zurück. Die Reihenfolge dieser Pixelwerte entspricht dabei genau Reihenfolge ihrer Speicherung.

V1.5 Sortieren

Ergänzen Sie die Methode `sort(float[] Array)` der Klasse `LocalNeighbours`. Diese nimmt die aktuellen Pixel, welche in der lokalen Nachbarschaft abgespeichert wurden und sortiert diese. Zum Sortieren setzen Sie den unten angegebenen Pseudo-Code um. Es genügt also nicht das Array zu sortieren, Sie sollen auch den angegebenen Sortieralgorithmus umsetzen.

```
sort(A)
beginn := -1
ende := Laenge(A)
solange beginn kleiner als ende wiederhole
    beginn := beginn + 1
    ende := ende - 1
    fuer (i = beginn, solange i < ende, i++) wiederhole
        falls A[i] > A[i + 1] dann
            vertausche(A[i], A[i+1])
        ende falls
    ende fuer
    fuer (i = ende, solange --i >= beginn) wiederhole
        falls A[i] > A[i + 1] dann
            vertausche(A[i], A[i+1])
        ende falls
    ende fuer
ende wiederhole solange
```

V1.6 Median

Ergänzen Sie die Methode `getMedian(float[] Array)` der Klasse `LocalNeighbours`. Diese soll den Median der aktuell gespeicherten Pixel der lokalen Nachbarschaft zurückgeben. Achten Sie bei Ihrer Implementierung darauf, dass der Median sowohl bei einer geraden als auch einer ungeraden Anzahl an Pixeln funktionieren soll.

V1.7 Medianfilter

Ergänzen Sie die Methode `nonLinearFilterMedian(Image image, int filtersize)`, die das nichtlineare Filtern mithilfe eines Medianfilters implementiert. Beachten Sie hierbei auch wieder die passende Randbehandlung! Verwenden Sie die Klasse `LocalNeighbours` und die von Ihnen ergänzten Methoden um den Medianfilter korrekt umzusetzen.

V2 Ticketverkauf

Implementieren Sie eine `public`-Methode

```
sellTickets(Collection<? extends TicketRequest> requests, int capacity)
```

mit Rückgabetyp `List<TicketRequest>`. Diese Methode erhält eine nach Eingangszeitpunkt sortierte Liste von Ticketanfragen. Die Ticketanfragen erhalten neben für diese Aufgabe irrelevanten Daten wie Kundenname etc. insbesondere den Status der Kundenanfrage, modelliert als `enum ReqType`. Aufgrund der sehr hohen Nachfrage werden drei Statuswerte unterschieden: `MEMBER` für Mitglieder, `SUB` (kurz für `subscription`) für Nicht-Mitglieder mit Dauerkarte sowie `REGULAR` für normale Kunden (weder Mitglieder noch Dauerkarteninhaber).

Für den Verkauf steht eine Anzahl `capacity` für die Tickets zur Verfügung. Der Verkauf soll wie folgt erfolgen:

1. Als erstes erhalten alle Mitglieder in der Reihenfolge der Liste ein Ticket, solange die Ticketanzahl noch nicht erreicht wurde.
2. Sind noch Tickets verfügbar, werden diese im zweiten Schritt an alle Dauerkarteninhaber vergeben, solange die Kapazität ausreicht.
3. Sind immer noch Tickets verfügbar, werden auch die normalen Kunden bis zur Kapazitätsgrenze bedient.

Als Ergebnis ist eine Liste mit allen verkauften Tickets zu liefern. Bitte beachten Sie, dass die Einträge der Liste nicht nach Status sortiert sind. Zusätzlich ist davon auszugehen, dass die Nachfrage das Ticketangebot (deutlich) übersteigt.

Verbindliche Anforderung: Die Eingabeliste darf nicht verändert werden! Insbesondere ist das Einfügen, Löschen einzelner Elemente oder das Sortieren der Liste verboten.

V3 Klassen und Interfaces

V3.1

Schreiben Sie ein `public`-Interface `X`, das eine Objektmethode `m1` mit Rückgabetyp `java.lang.Exception` hat, wobei `m1` einen `double`-Parameter `n` und einen `String`-Parameter `str` hat.

V3.2

Schreiben Sie ein `public`-Interface `Y`, das von `X` erbt und zusätzlich eine Klassenmethode `m2` ohne Parameter hat, die `double` zurückliefert.

V3.3

Schreiben Sie eine `public`-Klasse `XY`, die `X` und Methode `m1` implementiert. Konkret soll Methode `m1` eine Referenz auf ein Objekt vom Typ `NullPointerException` zurückliefern, das mit Konstruktor ohne Parameter eingerichtet wird. Klasse `XY` soll ein `public`-Attribut `p` vom Typ `char` haben sowie einen `public`-Konstruktor ohne Parameter. Der Konstruktor soll `p` auf den Wert 42 setzen. Weiter soll `XY` eine `protected`-Objektmethode `m3` mit

Rückgabe `true` oder `false` und Parameter `xy` vom Typ `XY` haben, aber nicht implementieren.

V3.4

Schreiben Sie eine `public`-Klasse `YZ`, die von `XY` erbt und sowohl `Y` als auch das Interface `java.util.Comparator<Integer>` implementiert. Die Methoden `m1` und `m2` sollen nicht in `YZ` überschrieben bzw. implementiert sein, und `m3` soll genau dann `true` zurückliefern, wenn der Wert `p` von `xy` gleich dem Wert `p` des eigenen Objektes ist. Der Konstruktor von `YZ` ist `public`, hat einen `long`-Parameter `r` und ruft den Konstruktor von `XY` auf. Methode `compare` von `Comparator<Integer>` hat bekanntlich zwei Parameter vom Typ `Integer` und liefert `int` zurück; in `YZ` soll sie `+1` zurückliefern, wenn der `int`-Wert im ersten Parameter um mindestens 5 höher als der `int`-Wert im zweiten Parameter ist, `+1`, wenn es genau umgekehrt ist, und `0` sonst.

V4 Mittelwert

Gegeben sei eine Klasse `X`, die generisch mit `T` parametrisiert ist.

Die `public`-Objektmethode `m1` hat ein Array `a` mit Komponententyp Array von `int` als Parameter und gibt ein Array von `double` zurück, das dieselbe Länge wie `a` haben soll. An jedem Index `i` von `a` soll das zurückgelieferte Array das arithmetische Mittel der Werte in `a[i]` enthalten.

Ihre Methode kann ohne Überprüfung davon ausgehen, dass der Parameter `a` nicht gleich `null` ist und dass `a` positive Länge hat.

Verbindliche Anforderung:

Rekursion darf nicht verwendet werden, das heißt, die Aufgaben müssen durch Schleifen realisiert werden. Dies müssen `for`-Schleifen sein. Zudem dürfen keine Klassen aus der Java-Standardbibliothek oder aus anderen Bibliotheken verwendet werden (Arrays fallen natürlich nicht darunter).

H Dreizehnte Hausübung

Fraktale

Gesamt 8 Punkte

H1 Visualisierung komplexer Zahlen

4 Punkte

In der Mathematik haben Sie die komplexen Zahlen kennengelernt. Diese bilden einen Körper, auf dem wir Operationen wie Addition und Multiplikation definieren können. Eine komplexe Zahl $z \in \mathbb{C}$ lässt sich dabei ausdrücken als:

$$z = a + bi \quad \text{mit } a, b \in \mathbb{R}$$

Dabei nennen wir a den realen und b den imaginären Anteil von z . In dieser Aufgabe betrachten wir nun die Folge:

$$z_{n+1} = z_n^2 + c \quad \text{mit } z, c \in \mathbb{C} \quad \text{und } z_0 = 0$$

Die Mandelbrot-Menge ist nun definiert als Menge aller $c \in \mathbb{C}$, für welche die Menge z_n beschränkt ist. Formal ausgedrückt ist die Mandelbrot-Menge also definiert als:

$$\mathcal{MB} = \{c \in \mathbb{C} \mid \exists b \in \mathbb{R} \forall n \in \mathbb{N} : |z_n| < b\}$$

Wir betrachten nun einen kleinen Ausschnitt der komplexen Zahlenebene und ordnen dabei jedem Pixel eine komplexe Zahl zu. In Abbildung 7 sehen Sie ein Beispiel für eine Mandelbrot-Menge.

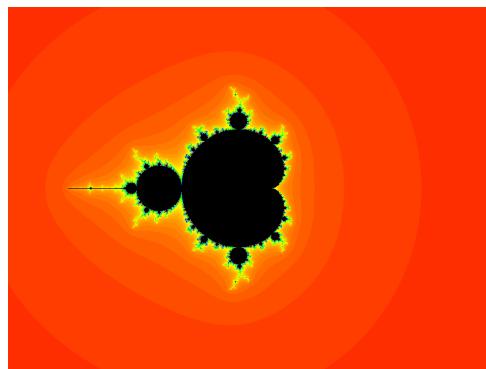


Abbildung 7: Visualisierung einer Mandelbrot-Menge

Zur Visualisierung dieser Zahlenebene iterieren wir über alle Bildpunkte. Dafür legen wir eine maximale Anzahl an Iterationen n_{\max} , sowie einen Radius $r_{\max} = b$ fest. Für jeden Pixel wird dann die oben genannte Folge maximal n_{\max} -mal iteriert. Die Folge ist genau dann beschränkt, wenn gilt:

$$\forall n \leq n_{\max} : |z_n| \leq r_{\max}$$

Besitzt ein Pixel eine beschränkte Folge, so wird dieser als schwarz eingefärbt. Für alle anderen Pixel wird das größte n gespeichert, für welches $|z_n| \leq r_{\max}$, um daraus später einen Farbwert zu bestimmen. Bildlich ist dies die Anzahl an Iterationen, in denen der Kreis mit Radius r_{\max} nicht verlassen wurde.

Zum Schluss wird diese Information noch in ein Farbschema überführt und einer Farbe zugeordnet. Im Pseudo-Code sieht dieses Vorgehen **pro Pixel** (x, y) folgendermaßen aus:

Algorithm 1 Generate Mandelbrot set

```

1: procedure GENERATE
2:    $z \leftarrow 0 + 0i$ 
3:    $c \leftarrow getComplexValueForPixel(x, y)$ 
4:    $i \leftarrow 0$ 
5:   loop
6:     if  $|z| \geq rMax$  or  $i \geq maxIter$  then break
7:      $z \leftarrow z \cdot z + c$ 
8:      $i ++$ 
9:   end loop
10:   $color \leftarrow colorMap(i)$ 
11:  setPixelColor( $x, y, color)$ 
```

Um die Werte für c zu bestimmen, beziehen Sie einen Zoomfaktor mit ein. Für den imaginären Teil der komplexen Zahl bilden wir die Differenz der aktuellen Pixelhöhe zur Hälfte der Bildhöhe und teilen diese Differenz durch den Zoomfaktor. Für den realen Teil machen wir das Äquivalente, nur mit der Breite.

Die Farbe bestimmen wir über den HSV-Farbraum. Dabei wählen wir:

$$\text{color} = \begin{cases} i/100 & = H \\ 1 & = S \\ i < n_{\max} ? 1 : 0 & = V \end{cases}$$

Zur Bearbeitung sind bereits folgende (teils unvollständige) Klassen im Package H1 gegeben:

- **ComplexNumber** dient zur Modellierung der komplexen Zahlen.
- **Set** ist die (fertig implementierte) Oberklasse der Klasse **MandelbrotSet**. Sie besitzt eine abstrakte Methode **generate**, sowie einige Attribute zur späteren Berechnung der Mandelbrot-Menge.
- **Visualizer** stellt die Menge später graphisch da. Diese Klasse ist bereits fertig und muss nicht von Ihnen bearbeitet werden.
- **Main** zum Aufruf der Methoden.

Haben Sie später alles implementiert, sollte der folgende Aufruf, welcher bereits in der Mainklasse zu finden ist, das Beispielbild in Abbildung 7 zurückliefern:

```
Set mb = new Mandelbrot();
new Visualizer(mb, 570, 150, 2 << 4).setVisible(true);
```

In den folgenden Teilaufgaben erhalten Sie weitere Informationen zur Implementierung.

H1.1 Komplexe Zahlen modellieren**1 Punkt**

Ergänzen Sie in der Klasse `ComplexNumber` folgende drei Methoden:

- `ComplexNumber add(ComplexNumber cn)`: addiert eine zweite komplexe Zahl mit der aktuellen und liefert das Ergebnis zurück
- `ComplexNumber mult(ComplexNumber cn)`: multipliziert eine zweite komplexe Zahl mit der aktuellen und liefert das Ergebnis zurück
- `double abs()`: liefert den Betrag der komplexen Zahl zurück

Die Rechenregeln finden Sie beispielsweise in der deutschsprachigen Wikipedia hier.

H1.2 Mandelbrot-Menge bestimmen**2 Punkte**

Ergänzen Sie nun in der Klasse `MandelbrotSet` die Methode `generate`. Diese bekommt die Höhe und Breite des zu erstellenden Bildes übergeben, sowie den Zoomfaktor, den Radius und die maximale Anzahl an Iterationen. Wenden Sie zur Berechnung den vorgestellten Algorithmus pro Pixel an und liefern Sie das fertige Bild zurück.

Streng genommen sind die Farbmodelle nicht identisch, Sie können zur Farbbestimmung allerdings die bereits vorimplementierten Methoden für den HSB-Farbraum der Klasse `Color` verwenden, siehe: [https://docs.oracle.com/javase/7/docs/api/java.awt/Color.html#getHSBColor\(float,%20float,%20float\)](https://docs.oracle.com/javase/7/docs/api/java.awt/Color.html#getHSBColor(float,%20float,%20float))

H1.3 Julia-Menge bestimmen**1 Punkt**

Zum Abschluss wollen wir uns eine zweite Menge, die Julia-Menge, anschauen. Diese steht in Beziehung zur Mandelbrot-Menge, da die Mandelbrot-Menge eine Beschreibungsmenge der Julia-Menge quadratischer Polynome ist. Sei nun wieder $z \in \mathbb{C}$ und sei p ein Polynom, dann ist die Julia-Menge dieses Polynoms definiert als: $z_{n+1} = p(z_n)$ mit $z \in \mathbb{C}$. Wir sehen also schnell, dass wir die Mandelbrot-Menge genau dann erhalten, wenn wir für p ein quadratisches Polynom mit linearem Glied gleich 0 wählen. In Abbildung 8 sehen Sie das Beispiel für die Visualisierung einer Julia-Menge.

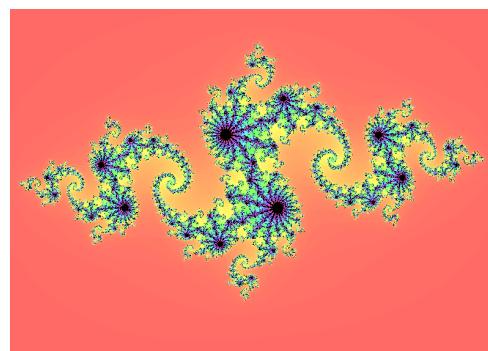


Abbildung 8: Visualisierung einer Julia-Menge

Der Unterschied der Generierung zwischen Julia- und Mandelbrotmenge liegt in der Iteration. Ihre Aufgabe ist es diesmal die Methode `generate` in der Klasse `JuliaSet` zu ergänzen. Dafür ist wieder ein Algorithmus angegeben.

Algorithm 2 Generate Julia set

```

1: procedure GENERATE
2:    $z \leftarrow chooseStartValue()$ 
3:    $c \leftarrow getComplexValueForPixel(x,y)$ 
4:    $i \leftarrow 0$ 
5:   loop
6:     if  $|c| \geq rMax$  or  $i \geq maxIter$  then break
7:      $newReal \leftarrow Re(c)^2 - Im(c)^2$ 
8:      $Im(c) \leftarrow 2 * Re(c) * Im(c)$ 
9:      $Re(c) \leftarrow newReal$ 
10:     $c \leftarrow c + z$ 
11:     $i++$ 
12:   end loop
13:    $color \leftarrow colorMap(i)$ 
14:   setPixelColor( $x, y, color$ )

```

Die Julia-Menge ist dabei abhängig vom gewählten Startwert von z . Um das Beispielbild zu erhalten wählen Sie $z_0 = 0.156i - 0.8$.

Das Farbschema für den HSV-Farbraum ändern wir hier minimal ab:

$$\text{color} = \begin{cases} (i \bmod 256)/255 &= H \\ 0.6 &= S \\ i < n_{\max} ? 1 : 0 &= V \end{cases}$$

Haben Sie alles implementiert, sollte der folgende Aufruf, welcher bereits in der Mainklasse zu finden ist, das Beispielbild in Abbildung 8 zurückliefern:

```
Set j = new Julia();
new Visualizer(j, 250, 250, 2).setVisible(true);
```

H1.4 Erkunden der Menge

Diese Aufgabe wird nicht bepunktet und stellt eine freiwillige Möglichkeit dar. Nach erfolgreicher Implementierung können Sie einmal am Zoomfaktor und den Werten für c herumspielen und so näher in die Mengen hereinzoomen. So erhalten Sie ganz neue Einblicke.

Funktioniert Ihre Implementierung der Julia-Menge, sodass Sie das Beispielbild erhalten, können Sie einmal verschiedene Startwerte z_0 ausprobieren. Hier einige Startwerte, welche einige kreative Mengen hervorrufen:

- $z_0 = -0.81i$
- $z_0 = 0.6i - 0.4$
- $z_0 = -0.3842i - 0.70176$

H2 Drachenkurve**2 Punkte**

Die sogenannte Drachenkurve ist eine fraktale Kurve, die aus wenigen Schritten konstruiert werden kann. Zur Erstellung dieser Kurve erstellen wir den Drachencode. Dieser Code gibt an, wie wir die Linien für die Drachenkurve zu zeichnen haben. Im Code selbst kodieren wir dabei durch einen String "**R**" eine Drehung um 90 Grad nach rechts und mit "**L**" entsprechend eine Drehung um 90 Grad nach links. Zwischen diesen Drehungen wird später eine Linie gezogen. Es gilt folgende Vorschrift:

- Der Drachencode 0-ter Ordnung ist leer, also "".
- Der Drachencode 1-ter Ordnung ist "**R**".
- Der Drachencode n-ter Ordnung ist der Drachencode (n-1)-ter Ordnung mit einem zusätzlichen "**R**" am Ende. An diesen Code wird dann nochmals der Drachencode (n-1)-ter Ordnung gehängt, wobei jedoch das mittlere Element durch ein "**L**" ersetzt wird.

Der gesamte Drachencode wird dann als einziger String, bestehend aus einer Sequenz von "**L**" und "**R**", kodiert.

H2.1 Drachencode erzeugen**1 Punkt**

Ergänzen Sie die Funktion `String getSequence(int n)` in der Klasse `DragonCurve`, welche den Drachencode n-ter Ordnung generiert und zurückliefert.

Verbindliche Anforderung: Ihre Implementierung muss rekursiv arbeiten, Sie dürfen also keine Schleifen verwenden.

H2.2 Drachenkurve zeichnen**1 Punkt**

Als nächstes wollen wir die Drachenkurve anhand des Drachencodes zeichnen. Für den Drachencode 14-ter Ordnung finden Sie die Drachenkurve in Abbildung 9.

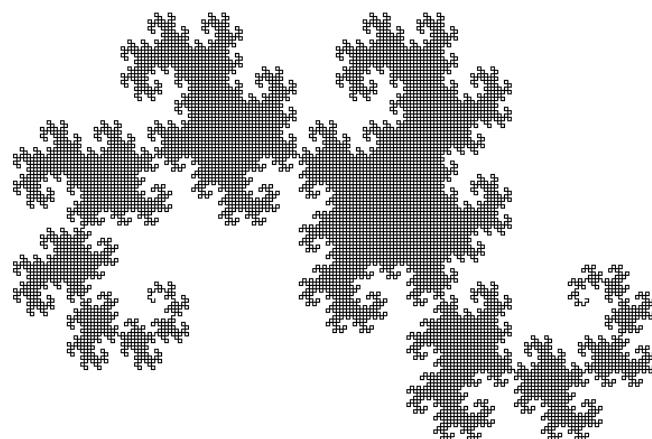


Abbildung 9: Drachenkurve für n=14

Ergänzen Sie die Methode `paint()` in der Klasse `Visualizer`. Die Klasse besitzt die `double`-Attribute `startingAngle` (der Winkel unter dem wir die erste Linie ziehen) und `len` (die Länge der einzelnen Linien), sowie das Attribut `turn` vom Typ `String`, welches den Drachencode aus der vorherigen Aufgabe darstellt.

Sie starten am Punkt (250, 375). Zeichnen Sie von diesem Startpunkt ausgehend eine Linie der Länge `len` unter dem Winkel `startingAngle`. Von dort an aktualisieren Sie den Winkel immer gemäß des Drachencodes, um 90-Grad Drehungen nach links oder rechts. Nach dieser Drehung zeichnen Sie wieder eine Linie der Länge `len` und wiederholen dies solange, bis Sie den gesamten Drachencode abgearbeitet haben.

Hinweis:

Eine Linie von (x_1, y_1) zu (x_2, y_2) realisieren Sie durch `g.drawLine(x1, y1, x2, y2)`.

H3 Sierpinski-Teppich

2 Punkte

Als letztes Fraktal werfen wir einen Blick auf den Sierpinski-Teppich. Gegeben sei ein Quadrat, von diesem entfernen wir in der Mitte genau ein Neuntel der Fläche (hier: schwarz färben). Nun verbleiben wieder 8 quadratische Flächen. In jeder dieser Flächen entfernen wir wieder ein Neuntel der Fläche in der Mitte und immer so weiter. Sei der Flächeninhalt zu Beginn $A_0 = 1$, dann beträgt der Flächeninhalt bzw. der Anteil an weißer Flächen nach $n + 1$ Iterationen:

$$A_{n+1} = A_n - \frac{8^n}{9^{n+1}}$$

Ein Beispiel für einen Sierpinski-Teppich nach 3 Iterationen finden Sie in Abbildung 10.

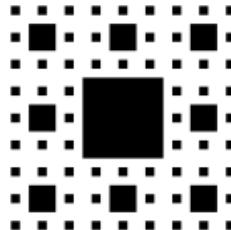


Abbildung 10: Sierpinski Teppich nach 3 Iterationen

Ergänzen Sie die Methode `paint` in der Klasse `SierpinskiCarpet`. In der Klasse wird bereits ein quadratisches Fenster der Länge `len` (gespeichert im gleichnamigen `private`-Attribut) erzeugt. Die Methode `paint` soll nun den Sierpinski-Teppich nach `n` (ebenfalls als `private`-Attribut verfügbar) Iterationen in dieses Fenster zeichnen.

Hinweis:

Über `g.fillRect(x, y, width, height)` erzeugen Sie ein gefülltes Rechteck, dessen oberer linker Punkt bei (x, y) liegt und die spezifizierte Länge und Breite besitzt.

Verbindliche Anforderung: Ihre Lösung muss rekursiv arbeiten, es sind also keine Schleifen erlaubt!

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.4

Übungsblatt 2

Themen: Funktionen höherer Ordnung, Lambda-Ausdrücke

Relevante Folien: Funktionale Abstraktion

Abgabe der Hausübung: 09.11.2018 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Theoriefragen

★ ★ ★

Erklären Sie kurz in eigenen Worten, die folgenden Konzepte:

1. Was sind Funktionen höherer Ordnung? Wo liegen ihre Vorteile?
2. Was ist ein Lambda-Ausdruck?
3. Wieso sollte man Abstraktion beim Programmieren verwenden?

V2 Ausdrücke mit Funktionen höherer Ordnung

★ ★ ★

In der Vorlesung haben Sie die Funktionen `my-map`, `my-filter` und `my-fold` kennengelernt und diese selbst implementiert. Alle drei Funktionen gibt es mit identischer Funktionalität bereits vordefiniert in DrRacket und haben die Namen `map`, `filter` und `foldr`.

Was liefern die folgenden Ausdrücke zurück? Arbeiten Sie hier ausschließlich mit Stift und Papier und verwenden Sie DrRacket erst hinterher, nur zur Überprüfung Ihrer Ergebnisse.

1. (`map + (list 1 2 3)(list 4 5 6)`)
2. (`filter positive? (list 1 -2 3 4 -5)`)
3. (`foldr + 0 (list 5 -9 3 2 5 6)`)
4. (`filter string? (list 1 2 "3" 4 "abc")`)
5. (`first (map list (list "x" "y" "z"))`)
6. (`map list (list "a" "b" "c")(list 1 2 3)(list true false true)`)
7. (`foldr cons (list -10 -1)(list 1 10 100 1000)`)
8. (`foldr list (list -10 -1)(list 1 10 100 1000)`)

V3 Funktionen höherer Ordnung verwenden

★ ☆ ☆

Definieren Sie die folgenden Funktionen. Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden.

- Eine Funktion `zip`, die aus zwei gleich langen Listen eine Liste von geordneten Paaren macht. Beispiel:
`(zip (list "a" "b") (list 1 2)) → (list (list "a" 1) (list "b" 2))`
- Eine Funktion `vec-mult`, die zwei gleich lange Listen von Zahlen erhält und das Skalarprodukt, also die Summe der paarweisen Produkte berechnet. Beispiel:
`(vec-mult (list 1 2 3) (list 4 5 6)) → (+ (* 1 4) (* 2 5) (* 3 6)) → 32`

V4 Lambda-Ausdrücke

★ ☆ ☆

```

1  ;;
2 (define (z x)
3  ;;
4      (lambda (y) (* x y)))

```

Ergänzen Sie den Vertrag, sowohl für die Funktion `z`, als auch für den Lambda-Ausdruck. Was liefert `((z 3) 4)` zurück?

V5 Foo Reloaded I

★ ★ ☆

Erinnern Sie sich noch an die Funktion `foo` aus Aufgabe V7 vom letzten Übungsblatt? Zur Erinnerung: Gegeben ist ein Struct-Typ `abc` mit zwei Feldern `a` und `b`. Die Funktion `foo` bekommt einen Parameter `p` und liefert falls `p` vom Typ `abc` und zudem der Wert im Feld `b` von `p` eine Liste ist eine Liste zurück, deren erstes Element der Wert von Feld `a` in `p` ist, und der Rest der zurückgelieferten Liste ist die Liste im Feld `b` von `p` (also eine Liste in der Liste). Andernfalls liefert `foo` einfach `false` zurück.

Definieren Sie nun eine Funktion `bar1`, die einen Parameter `lst` übergeben bekommt. Für jedes Element `x` in `lst`, das vom Typ `abc` ist, soll die Ergebnisliste von `bar1` das Ergebnis der Anwendung von `foo` auf `x` enthalten. Weitere Elemente darf die Ergebnisliste von `bar1` nicht enthalten.

Verbindliche Anforderung: Sie dürfen in dieser Aufgabe noch keine Funktionen höherer Ordnung wie `map` oder `filter` verwenden. Diese Funktionalitäten müssen von Ihnen selbst implementiert werden.

V6 Foo Reloaded II

★ ★ ☆

Definieren Sie nun eine Funktion `bar2`. Diese besitzt die gleiche Funktionalität wie `bar1` aus Aufgabe V5. In dieser Aufgabe wird die Funktionalität allerdings nicht mehr selbst geschrieben, sondern an die vordefinierten Funktionen `map` und `filter` delegiert. Nutzen Sie Lambda-Ausdrücke, welche Sie innerhalb der Aufrufe von `map` und `filter` definieren.

V7 Kartesisches Produkt

★ ★ ☆

Definieren Sie eine Funktion `cartesian-prod`, die zwei Zahlenlisten erhält und das kartesische Produkt der beiden bildet. Beispiel:

```
(cartesian-prod (list 1 2)(list 3 4))
→ (list (list 1 3)(list 1 4)(list 2 3)(list 2 4))
```

Verwenden Sie eine Kombination aus Funktionen höherer Ordnung und Lambda-Ausdrücke für Ihre Lösung.

V8 Bibliothek Leihgebühren

★ ★ ☆

Eine Bibliothek verwaltet ihr Leihsystem nun in Racket. Dazu wird ein neuer Struct-Typ `br` definiert.

```
(define-struct br (id pop type))
```

Das Feld `id` ist dabei ein String und stellt die ID-Nummer des ausgeliehenen Buches dar. Das Feld `pop` ist eine Zahl zwischen 1 bis 6 und gibt die Beliebtheit des Buches an (je größer die Zahl, desto beliebter das Buch). Das letzte Feld `type` ist wieder ein String, der entweder "Single" oder "Subscription" sein kann, je nachdem ob es sich um eine einmalige Ausleihe oder einen Abonnenten handelt.

Folgende Regeln gelten in der Bibliothek: Abonnenten zahlen für jedes ausgeliehene Buch pauschal 1,50€. Bei normalen Kunden berechnet sich der Preis über die Beliebtheit des Buches. Pro Beliebtheitsstufe kostet das Buch 1,75€. Somit kostet ein Buch mit Beliebtheitsstufe 3 beispielsweise 5,25€.

Ihre Aufgabe ist es nun eine Funktion `fee-total` zu definieren. Diese enthält eine Liste von `br`-Structs (die Ausleihliste) und gibt die Gesamteinnahmen aus eben dieser Ausleihliste zurück.

V9 Wer bekommt die Zulassung?

★ ★ ☆

Schreiben Sie eine Prozedur zur Prüfung der Zuteilung einer Studienleistung im Modul X. Dort sind 50 Hausaufgaben-, 35 Zwischenklausur- und 50 Projektpunkte sowie insgesamt mindestens 180 Punkte aus den drei Bereichen zusammen erforderlich. Definieren Sie dazu eine Funktion `passed` mit folgender Signatur

```
(list of number)(list of (list of number number number))-> (list of number)
```

Diese Funktion erhält aus Datenschutzgründen separat die Liste der Matrikelnummern sowie eine Liste von Listen mit Punkten für Hausaufgaben, Zwischenklausur und Projekt (in dieser Reihenfolge). Die Precondition ist dabei, dass die Listen gleich lang sind und dass die Matrikelnummer an Position i der ersten Liste zu den Punkten an Position i der zweiten Liste gehört (vergessen Sie die Precondition nicht im Vertrag der Funktion). Die Ergebnisliste enthält die Matrikelnummern aller Studierenden, die die Bedingungen für die Studienleistung erfüllt haben. Die Reihenfolge der Studierenden soll dabei erhalten bleiben.

V10 Bildverarbeitung in Racket



Funktionen aus dieser Aufgabe können auch für Aufgabe H verwendet werden.

Um diese Aufgaben in DrRacket ausführen zu können, setzen Sie bitte (`require 2htdp/image`) in die oberste Zeile Ihrer Datei ein (vergleichen Sie auch die Vorlage der Hausübung).

Bilder bestehen aus vielen aufeinanderfolgenden Pixeln. Jedes Pixel nimmt dabei genau die Farbe an, die durch sein sogenanntes RGB-Tripel beschrieben werden. Dies ist durch die Darstellung im sogenannten RGB-Farbraum, ein sogenannter technischer Farbraum, der die Farbwahrnehmung durch das additive Mischen der drei Grundfarben nachbildet, begründet. Jede Farbe lässt sich dabei durch ein Tripel (R, G, B) darstellen, wobei die drei Zahlen jeweils den Anteil der jeweiligen Grundfarbe angeben. So ist das klassische rot durch $(255, 0, 0)$, gelb als Mischung zweier Grundfarben durch $(255, 255, 0)$ und braun als Mischung aller Grundfarben als $(153, 102, 51)$ dargestellt.

Der Einfachheit wegen benutzen wir nur Bilder im PNG-Format (dh. Dateiendung `.png`), die keine transparenten Farben enthalten, also keinen Alphakanal besitzen. Ein Bild ist in Racket immer ein Struct vom Typ `image`. Jedes Bild besteht aus seinen aufeinanderfolgenden Pixeln. In Racket ist ein Pixel als `color`-Struct definiert:

```
(define-struct color (red green blue alpha))
```

Die ersten drei Felder sind das Tripel des RGB-Farbraums und liegen zwischen 0 und 255. Den Alpha-Wert ignorieren wir in dieser Übung, er soll immer auf 255 gesetzt werden. Folgende Funktionen gibt es bereits für die Bildverarbeitung in Racket:

- Um aus einem `image` die entsprechenden `color`-Structs zu bekommen, gibt es die Funktion `(image->color-list img)`. Diese gibt eine Liste von `color`-Structs für das übergebene Bild zurück.
- Um aus einer Liste von `color`-Structs ein Bild zu generieren gibt es die Funktion `(color-list->bitmap clr-lst width height)`. Diese benötigt neben der Liste von `color`-Structs auch die Breite und Höhe des zu generierenden Bildes (über die Funktionen `(image-width img)` und `(image-height img)` abrufbar).
- Mit `(bitmap/file "image.png")` laden Sie das Bild im PNG-Format namens “image“, welches im gleichen Verzeichnis wie die `.rkt` Datei liegt. Mittels `(save-image img "out.png")` speichern Sie ein `image`-Struct unter dem Namen “out“ dort.

Nutzen Sie für die folgenden beiden Aufgaben Funktionen höherer Ordnung!

1. Definieren Sie eine Funktion `(count-black-white img)`. Diese bekommt ein Bild übergeben, welches nur aus schwarzen $(0,0,0)$ und weißen Pixeln $(255,255,255)$ besteht. Zurückgegeben werden soll eine zweielementige Liste, welche an erster Position die Anzahl an schwarzen und an zweiter Position die Anzahl an weißen Pixeln enthält.
2. Definieren Sie eine Funktion `(negative-transformation img)`. Diese bekommt ein Bild als `image`-Struct übergeben und gibt die Negativtransformation dieses Bildes zurück. Dazu berechnen Sie die RGB-Werte für jeden Pixel neu über den folgenden Zusammenhang: $(R_{\text{neg}}, G_{\text{neg}}, B_{\text{neg}}) = (255 - R, 255 - G, 255 - B)$

H Zweite Hausübung Protanopie

Gesamt 10 Punkte

In dieser zweiten Hausübung beschäftigen Sie sich mit der sogenannten Protanopie, der Rot-Grün-Blindheit. Darunter versteht man eine Anomalie der Netzhaut, die zur Farbfehlsichtigkeit führt und zur Folge hat, dass Rottöne nicht mehr wahrgenommen werden können.

Die Netzhaut des menschlichen Auges besteht aus zwei Arten von Sinneszellen - die Stäbchen und die Zapfen. In dieser Hausübung beschäftigen wir uns nur mit den Zapfen der Netzhaut, da diese für das Farbsehen bei Tageslicht verantwortlich sind. Wir unterscheiden hierbei drei verschiedene Typen von Zapfen:

- S-Zapfen (Short wavelength/kurze Wellenlänge): Decken den blauen Bereich des sichtbaren Farbspektrums ab.
- M-Zapfen (Medium wavelength/mittlere Wellenlänge): Decken einen Bereich zwischen blauem und orangem Licht ab.
- L-Zapfen (Long wavelength/lange Wellenlänge): Decken die Wahrnehmung von rotem Licht ab.

Bei Menschen mit Protanopie sind entweder keine L-Zapfen auf der Netzhaut vorhanden oder die vorhandenen L-Zapfen liefern eine falsche Farbantwort. Dies hat zur Folge, dass Menschen mit Protanopie im kurzweligen Bereich (wie Farbgesunde) ein sattes Blau, im mittelwelligen Bereich Grau und im langwelligen Bereich ein sattes Gelb sehen. Ihr Ziel ist es, in dieser Aufgabe die Protanopie zu simulieren. Dafür finden Sie Beispiele in Abbildung 1.



Abbildung 1: Beispielbilder vor der Simulation



Abbildung 2: Beispielbilder nach angewandter Simulation der Protanopie

Bildquellen: Äpfel, Bahn und Regenbogen.

Um diese Simulation zu bewerkstelligen, werden Sie mit zwei verschiedenen Farträumen arbeiten. Der erste ist der sehr populäre RGB-Farbraum, den Sie bereits in den vorbereitenden Übungen kennengelernt haben. Der zweite Farbraum ist der LMS-Farbraum. Dieser erhält auch ein Tripel (L, M, S) , wobei die drei Einträge an die menschlichen Zapfen angelehnt sind (vergleiche oben). Weitere Details sind für diese Übung nicht notwendig. Wir werden Sie in den nachfolgenden Aufgaben Schritt für Schritt anleiten.

Achtung: Verwenden Sie in allen Teilaufgaben Funktionen höherer Ordnung und Lambda-Ausdrücke, wo immer das möglich und sinnvoll ist. Falls Sie Funktionen von Grund auf selbst implementieren, die Sie mit Funktionen höherer Ordnung und Lambda-Ausdrücken kompakter hätten ausdrücken können, müssen Sie mit Punktabzügen rechnen. Ist die Verwendung explizit gefordert, verlieren Sie auf jeden Fall Punkte, sollten Sie diese Anforderung ignorieren.

Matrixarithmetik

Für die hier notwendigen Berechnungen werden wir Vektoren und Matrizen verwenden, wie Sie sie schon aus der Schulmathematik kennen sollten. Um diese in Racket darzustellen, verwenden wir verschachtelte Listen.

Für die Matrix $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ schreiben wir `(list (list 1 2 3)(list 4 5 6))`.

Für den Vektor $\begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$ schreiben wir `(list (list 7)(list 8)(list 9))`.

Sollten Sie mit der Matrizenmultiplikation nicht vertraut sein, so holen Sie dies an dieser Stelle nach. Ein gutes Beispiel findet unter folgendem Link:

https://mathepedia.de/Multiplikation_Definitionen_und_Operationen.html

In Worten bedeutet eine Matrizenmultiplikation folgendes: Um den Eintrag der Ergebnismatrix in Reihe i und Spalte j zu bekommen, so nehmen Sie die Reihe i der linken Matrix, die Spalte j der rechten Matrix und betrachten diese beide als Vektoren. Das Skalarprodukt der beiden ist dann der gesuchte Eintrag.

H1 Matrizenmultiplikation

3 Punkte

Definieren Sie zu Beginn eine Funktion `(matrix-multiplication m1 m2)`.

Diese erhält zwei Matrizen im zuvor beschriebenen Format und liefert die Ergebnismatrix im gleichen Format zurück.

Verbindliche Anforderung: Verwenden Sie für die Berechnung der Ergebnismatrix das in Aufgabe V3 vorgestellte Skalarprodukt an geeigneter Stelle. Deligieren Sie die Berechnung des Skalarproduktes an die Funktionen `foldr` und `map`.

Tipp: In dieser Aufgabe könnte es hilfreich sein zusätzliche Funktionen zu definieren, die Sie für die Berechnung verwenden wollen.

H2 Farbe zu Vektor vice versa**2 Punkte**

Definieren Sie in dieser Aufgabe zwei Funktionen.

1. Die Funktion (`color->vector clr`) bekommt ein `color`-Struct übergeben und gibt die RGB-Werte als Vektor zurück (also als Liste von Listen von Zahlen).
2. Die Funktion (`vector->color vec`) bekommt einen Vektor übergeben und gibt die Farbe als `color`-Struct zurück. Runden Sie die RGB-Werte immer auf ganze Zahlen herunter. Sollten Werte kleiner als 0 oder größer als 255 auftreten, so setzen Sie diese Werte einfach auf 0 beziehungsweise 255.

Hinweis: Beachten Sie für diese und die nachfolgenden Übungen nochmal besonders die Hinweise in Aufgabe V10.

H3 Simulation der Protanopie-Werte für einen Vektor 1 Punkt

In dieser Aufgabe wollen wir die Simulation der Protanopie an einem gegebenen RGB-Vektor durchführen¹. Definieren Sie dazu die Funktion

`(rgb-vector-protanopia vec)`

die einen RGB-Vektor bekommt und den Vektor mit den transformierten Werten zurückgibt.

Dazu transformieren Sie den gegebenen RGB-Vektor zunächst in den LMS-Farbraum:

$$\begin{pmatrix} L \\ M \\ S \end{pmatrix} = \begin{pmatrix} 17.8824 & 43.5161 & 4.1193 \\ 3.4557 & 27.1554 & 3.8671 \\ 0.02996 & 0.18431 & 1.4670 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Dann simulieren Sie die Protanopie im LMS-Farbraum:

$$\begin{pmatrix} L_{\text{Protanopie}} \\ M_{\text{Protanopie}} \\ S_{\text{Protanopie}} \end{pmatrix} = \begin{pmatrix} 0 & 2.02344 & -2.52581 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} L \\ M \\ S \end{pmatrix}$$

Zum Schluss wandeln Sie die simulierten Werte wieder in den RGB-Farbraum um:

$$\begin{pmatrix} R_{\text{Protanopie}} \\ G_{\text{Protanopie}} \\ B_{\text{Protanopie}} \end{pmatrix} = \begin{pmatrix} 0.0809 & -0.1305 & 0.1167 \\ -0.0102 & 0.0540 & -0.1136 \\ -0.0003 & -0.0041 & 0.6935 \end{pmatrix} \cdot \begin{pmatrix} L_{\text{Protanopie}} \\ M_{\text{Protanopie}} \\ S_{\text{Protanopie}} \end{pmatrix}$$

Alle benötigten Matrizen sind bereits in der Vorlage definiert.

¹Als Grundlage für die Berechnungen dieser Hausübung wurde die hier verlinkte Quelle genutzt.

Für die Aufgaben H4-H6 brauchen Sie keine eigenen Tests zu schreiben.

H4 Protanopie-Simulation für ein gegebenes Bild I 1 Punkt

Definieren Sie die Funktion (`protanopia-recursive img`), die ein Bild übergeben bekommt und die Protanopie-Simulation rekursiv auf jedes Pixel anwendet und das daraus resultierende Bild (`image-Struct`) zurückgibt.

Verbindliche Anforderung: Sie dürfen in dieser Aufgabe keine Funktionen höherer Ordnung verwenden, sondern nur “ganz normale” Rekursion. Definieren Sie sich geeignete, zusätzliche Funktionen zur Hilfe.

H5 Protanopie-Simulation für ein gegebenes Bild II 1 Punkt

Definieren Sie die Funktion (`protanopia-map img`), die ein Bild übergeben bekommt und die Protanopie-Simulation auf jedes Pixel, mittels Funktionen höherer Ordnung, anwendet. Kommt Ihnen bekannt vor?

Verbindliche Anforderung: Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden.

H6 Ähnlichkeit zweier Bilder 2 Punkte

Zum Abschluss wollen wir nun feststellen, wie viele Pixel eines Ausgangsbildes durch die Protanopie-Simulation gar nicht oder nur minimal modifiziert wurden. Dazu bestimmen wir die Anzahl an denjenigen Pixeln des Ausgangsbildes, die eine kleine Differenz zwischen dem jeweilig zugehörigen Pixel des Bildes mit angewandter Protanopie-Simulation aufweisen. Die Differenz zwischen zwei RGB-Werten, berechnen Sie über den folgenden Zusammenhang:

$$\text{Diff}_{(R_1, G_1, B_1), (R_2, G_2, B_2)} = |R_1 - R_2| + |G_1 - G_2| + |B_1 - B_2|$$

Anschließend teilen wir diese Anzahl durch die Gesamtanzahl an Pixeln im Ausgangsbild und multiplizieren das daraus resultierende Ergebnis mit 100, um den prozentualen Anteil an Pixeln zu erhalten, die gar nicht oder nur minimal modifiziert wurden. Definieren Sie dazu die folgende Funktion (`image-similarity img1 img2 max-difference`), die zwei gleichgroße Bilder und eine Zahl übergeben bekommt und zuerst die Differenzen zwischen allen RGB-Werten der beiden Bilder berechnet und eine Liste aus ihnen konstruiert. Anschließend entfernen Sie alle Differenzen aus eben dieser Liste, die größer als die übergebene Zahl `max-difference` ist. Nun teilen Sie die Anzahl an Elementen in dieser Liste durch die Gesamtanzahl an Pixeln im Ausgangsbild und multiplizieren das daraus resultierende Ergebnis mit 100, um den prozentualen Anteil an Pixeln zurückzugeben, die gar nicht oder nur minimal modifiziert wurden.

Verbindliche Anforderung: Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden. Realisieren Sie die Differenzfunktion als Lambda-

Ausdruck und definieren Sie das Filterprädikat innerhalb der Aufrufe von `map` bzw. `filter` unter Verwendung von `lambda`-Ausdrücken.

H7 Protanopie-Simulation auf Bildern

0 Punkte

Mithilfe der Vorlage können Sie nun die Protanopie-Simulation auf Bilder anwenden. Nutzen Sie die im Bildverarbeitungsabschnitt genannten “nützlichen“ Funktionen um ein Bild einzulesen, die Simulation durchzuführen und anschließend abspeichern zu können.

In der Vorlage zur Hausübung finden Sie bereits einige Bilder aus Abbildung 1, welche Sie dafür nutzen können.

Hinweise zur Bearbeitung:

Die Aufgaben bauen hier teilweise aufeinander auf. Für die korrekte Ausführung von Aufgabe H3 wird beispielsweise die Implementierung von Aufgabe H1 vorausgesetzt.

Bearbeiten Sie die Aufgabe dann trotzdem und tun Sie so, als hätten Sie die hervorgehenden Aufgaben korrekt lösen können. Wir werden in diesen Fällen Ihre fehlerhafte oder fehlende Lösung durch die Referenzlösung ersetzen. Sollte dann Aufgabe H3 korrekt implementiert sein und die Ausführung nur an Aufgabe H1 scheitern erhalten Sie trotzdem für diese Aufgabe dann volle Punkte.

Erinnerung an verbindliche Anforderungen: Für alle Racket-Hausübungen gilt, dass Zuweisung mittels `set!`, `let`, `begin` oder ähnliches nicht erlaubt ist! Sollten Sie durch eigene Recherche auf dieses Konstrukt stoßen dürfen Sie es nicht verwenden, Sie müssen die Hausübungen mittels normaler Rekursion, wie in der Vorlesung vorgestellt, lösen.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.2

Übungsblatt 3

Themen: Einführung Java, Eclipse und KarelJ

Relevante Folien: KarelJ

Abgabe der Hausübung: 16.11.2018 bis 23:55 Uhr

V Vorbereitende Übungen

Dieses Übungsblatt legt den Grundstein für alle weiteren Java Übungsblätter. Sie ist damit das exakte Pendant zu Übungsblatt 0 für Racket. Auch hier erwarten wir, dass Sie sich intensiv mit dem Blatt beschäftigen, da wir alle hier beschriebenen Formalitäten auf allen weiteren Übungsblättern als gegeben voraussetzen.

V1 Entwicklungsumgebung Eclipse installieren

In dieser ersten vorbereitenden Übung, gehen wir mit Ihnen Schritt für Schritt die Installation von Java und der Entwicklungsumgebung Eclipse durch.

V1.1 JDK installieren

Zu Beginn installieren wir das sogenannte Java Developement Kit (kurz: JDK). Wir verwenden in dieser Veranstaltung Java 8. Bevor Sie mit der Installation beginnen, können Sie überprüfen, ob Sie schon eine Java Version installiert haben und wenn ja, welche. Öffnen Sie dafür Ihre Konsole und geben Sie den Befehl `java -version` ein. Wenn noch keine JDK installiert ist, können Sie diese unter folgendem Link herunterladen:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Dabei ist darauf zu achten, das Java Development Kit (JDK) (mit Übersetzern etc. für die Programmierung) herunterzuladen, nicht die Laufzeitumgebung Runtime Environment (JRE) - mit dieser können keine selbstgeschriebenen Programme übersetzt werden! Das JRE ist bei der Installation des JDK bereits enthalten.

V1.2 Eclipse installieren

Wir verwenden die *Eclipse IDE for Java Developers*. Die aktuelle Version ist Eclipse SimRel 2018-09 und lässt sich hier herunterladen:

<https://www.eclipse.org/downloads/>

Die Installation von Eclipse setzt eine Java-Installation voraus. Auf den Poolrechnern ist Eclipse bereits installiert und mit dem Befehl `eclipse` & aufrufbar.

Für die Installation von Eclipse findet sich unter folgendem Link eine sehr hilfreiche Bilderstrecke:

<https://www.eclipse.org/downloads/packages/installer>

V2 Eclipse startklar machen

V2.1 Importieren von Vorlagen

Von uns bereitgestellte Vorlagen in moodle lassen sich in Eclipse leicht importieren. Laden Sie dazu die Vorlage herunter und speichern Sie sie an einem beliebigen Ort. Machen Sie dann einen Rechtsklick im *Package Explorer* von Eclipse und wählen *Import* aus. Wählen Sie dann *General*, dann *Existing Projects into Workspace*, dann *Select archive file* und dann *Browse*. Wählen Sie nun die heruntergeladene Vorlage im zip-Format aus und drücken Sie *Finish*.

V2.2 Projekt ausführen

Um nun das importierte Projekt auszuführen, öffnen Sie die Klasse (eine .java Datei in der Ordnerstruktur), in der die Methode `public static void main(String[] args)` zu finden ist (wird von uns angegeben in den Hausübungen). Diese dient als Einstiegspunkt. Mit einem Rechtsklick klicken Sie auf diese Klasse im *Package Explorer* und wählen *Run as* und dann *Java Application* aus, um die Klasse am Einstiegspunkt auszuführen. Das Drücken des grünen Play-Buttons in der oberen Leiste führt die zuletzt ausgewählte Klasse aus, die einen Einstiegspunkt besessen hat (dabei wird der gesamte Workspace betrachtet, also auch andere Projekte). Das heißt besitzt die Klasse, die aktuell im Eclipse-Editor offen ist, einen Einstiegspunkt, so wird diese beim Drücken ausgeführt. Andernfalls wird die Klasse ausgeführt, die zuletzt im Editor offen war und einen Einstiegspunkt hat.

V2.3 Treffpunkt und Fingerübung

Sie haben damit alle notwendigen Schritte zur Bearbeitung der Übungen kennengelernt. Für zusätzliche Einführungen zum Thema Eclipse besuchen Sie den entsprechenden Treffpunkt und bearbeiten Sie Fingerübung 3.

V2.4 KarelJ einbinden

Sie müssen sich nicht um das Einbinden von KarelJ bei den Hausübungen kümmern. In den Vorlagen, die wir Ihnen zur Verfügung stellen, ist dies bereits erledigt und Sie können den Code einfach ausführen.

V3 Ist Eclipse startbereit?



Importieren Sie die Vorlage V03 aus moodle und führen Sie die Klasse `EclipseReady.java` aus. Wenn Sie alles korrekt installiert haben, wird Ihnen in der Konsole unten Ihre installierte Java Version ausgegeben.

V4 Erste Schritte mit Karel



Öffnen Sie nun die Klasse `FirstStepsKarel.java`. Dort finden Sie eine Stelle, welche mit `TODO` gekennzeichnet ist. Fügen Sie hier Ihren Code ein, der folgendes umsetzt:

1. Erstellen Sie einen Roboter namens `karel`, der auf der Position (5,5) steht und nach Osten blickt. Er besitzt zu Beginn drei Beeper in seiner Tasche.
2. Lassen Sie `karel` nun zwei Schritte nach vorne laufen.
3. Drehen Sie `karel` nun so, dass er nach Süden blickt.
4. Lassen Sie `karel` einen Schritt nach vorne laufen.
5. Legen Sie einen Beeper von `karel` ab.
6. Lassen Sie `karel` zwei Schritte nach vorne laufen.
7. Legen Sie zwei Beeper mit `karel` ab.
8. Drehen Sie `karel` nun so, dass er nach Westen blickt.
9. Lassen Sie `karel` zwei Schritte nach vorne laufen.
10. Lassen Sie `karel` den Beeper aufheben.
11. Lassen Sie `karel` einen Schritt nach vorne laufen.

V5 Quadrat



Öffnen Sie nun die Klasse `Square.java`. Dort finden Sie eine Stelle, welche mit `TODO` gekennzeichnet ist. Fügen Sie hier Ihren Code ein, der folgendes umsetzt:

Zu Beginn platzieren Sie zwei Roboter in der Welt, von denen beide 20 Beeper besitzen. Der erste Roboter befindet sich in Position (1,1) und blickt nach Osten, der andere befindet sich in Position (10,10) und blickt nach Westen. Ihre Aufgabe ist es nun, ein (nicht ausgefülltes) Quadrat mithilfe der beiden Roboter zu zeichnen. Dabei soll sich am Ende des Programms jeder Roboter im Startpunkt des jeweils anderen befinden. In Abbildung 1 finden Sie einen Vorher-Nachher-Vergleich dieser Situation.

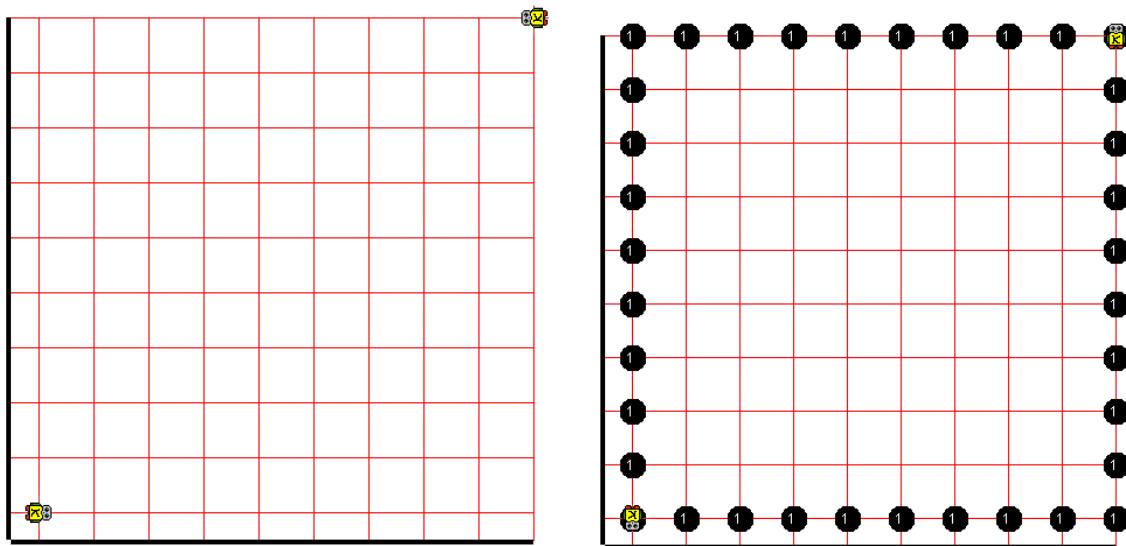


Abbildung 1: Vorher-Nachher-Vergleich

Verbindliche Anforderung: Das Laufen und Ablegen von Beepern darf nur innerhalb einer Schleife umgesetzt werden, in der in jedem Durchlauf jeder der Roboter genau einen Beeper ablegt! Lediglich das Drehen der Roboter darf außerhalb einer Schleife geschehen.

H Dritte Hausübung

Abgabe von Java Hausübungen

Gesamt 1 Punkt

Auch für alle Java-Hausübungen geben wir Ihnen Codevorlagen vor, die Sie unbedingt benutzen müssen! Wie Sie diese Vorlagen korrekt importieren können, sehen Sie nochmal in Aufgabe V2.1.

Auch für die Hausübungen in Java gibt es Namenskonventionen. Diese sind identisch zu denen in Racket, es unterscheidet sich lediglich das Dateiformat, welches Sie abzugeben haben. Für jedes Übungsblatt geben Sie ein eigenes Java-Projekt ab mit der identischen Namenskonvention wie zuvor, also: `Hnr_ln_fn`. Ein Projekt für die Abgabe dieser Hausübung kann also z.B. **H03_Mustermann_Max** heißen.

Nach dem Herunterladen der Vorlage wählen Sie die Vorlage im Package Explorer mit einem Rechtsklick aus und drücken erst *Refactor*, dann *Rename* um das Projekt nach der vorgegebenen Namenskonvention zu benennen.

Zur Abgabe exportieren Sie bitte Ihr gesamtes Projekt, indem Sie das Projekt im Package Explorer mit Rechts anklicken und *Export*, dann *General* und abschließend *Archive File* wählen. Wählen Sie alle Inhalte Ihres Projekts aus und benennen Sie das Archiv wie das Projekt, nur mit Endung `.zip`. Achten Sie darauf, dass die Optionen *Save in zip format* sowie *Compress the contents of the file* selektiert sind. Geben Sie am Ende genau diese zip-Datei ab. Nutzer anderer IDEs als Eclipse müssen darauf achten, dass sich im zip-Verzeichnis der Projektordner befindet und alle Java-Sourcen im Unterverzeichnis `src` stehen, Eclipse macht das automatisch.

Achtung: Wenn Sie eine der oben beschriebenen Anforderungen zur Abgabe verletzen, verlieren Sie Punkte! Sind Sie sich unsicher, nutzen Sie das Forum um nachzufragen.

Hinweis zu Namen mit Umlauten:

Sollte Ihr Vor- oder Nachname ein Sonderzeichen enthalten, so kann es sein, dass Sie in Eclipse den Fehler `Error: Could not find or load main class Klassenname` erhalten.

Sollten Sie also einen Umlaut in Ihrem Namen besitzen, so ersetzen Sie diesen am Besten (ä zu ae, ö zu oe und ü zu ue). Bei der Abgabe der Hausübung kann beim Exportieren des `.zip`-Ordners dann wie gewohnt mit Umlauten gearbeitet werden, dies bleibt Ihnen überlassen.

Wir bedanken uns bei Fabian Meerkötter für die Einsendung dieses Hinweises!

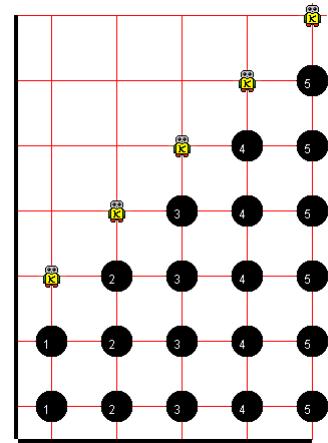
H1 Kaskade**1 Punkt**

In dieser dritten Hausübung sollen Sie mithilfe von 5 Robotern eine Kaskade zeichnen. Dabei sind die Roboter durchnummeriert von 1 bis 5. Für einen Roboter mit Nummer i gilt: Er steht an Position $(1, i)$ und blickt nach Norden. Jetzt soll er insgesamt $i + 1$ Schritte nach vorne gehen und vor jedem Schritt i Beeper ablegen.

Sie sehen in der Abbildung rechts das Endresultat nach dem Ausführen des beschriebenen Programms. Ergänzen Sie den Code an der mit `TODO` markierten Stelle so, um genau dieses Endresultat zu erhalten. Überlegen Sie sich zunächst wie viele Beeper jeder Roboter in seiner Tasche haben muss. Wenn die Roboter am Ende angelangt sind dürfen keine Beeper mehr in der Tasche übrig sein!

Verbindliche Anforderung: Sie müssen ineinander geschachtelte Schleifen verwenden.

Achtung: Sie erhalten diesen Punkt nur, wenn das Programm funktioniert und Sie sich an alle Abgabeformalitäten gehalten haben.



Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.1

Übungsblatt 4

Themen: KarelJ I

Relevante Folien: KarelJ

Abgabe der Hausübung: 23.11.2018 bis 23:55 Uhr

V Vorbereitende Übungen

V1 KarelJ



Beschreiben Sie kurz in ihren eigenen Worten worum es sich bei KarelJ handelt, wie die Welt aufgebaut ist und welche Grundfunktionen jeder Roboter beherrscht.

Für alle nachfolgenden Übungen: In der Abschlussklausur werden Sie keine Hilfsmittel zur Verfügung haben. Üben Sie also schon zu Beginn auch ohne Entwicklungsumgebung und nur mit Stift auf einem Blatt Papier zu programmieren.

V2 Liegen geblieben



Betrachten Sie den folgenden Codeausschnitt/führen Sie ihn selbst einmal aus:

```
1 Robot karel = new Robot(2,4,North,0);  
2 karel.move();  
3 karel.turnLeft();  
4 karel.putBeeper();
```

In welcher Zeile kommt es zu einem Problem und wieso?

V3 Rechteck



Schreiben Sie ein Programm, welches zwei Roboter `karel` und `larel` erstellt. Dabei soll `karel` mit Beepern ein Rechteck der Höhe 5 und der Breite 3 zeichnen. Nachdem das Rechteck gezeichnet wurde, soll `larel` alle Beeper wieder einsammeln. Überlegen Sie sich, wie Sie das Programm mit nur einer Schleife pro Roboter gestalten können. Orientieren Sie sich zur Hilfe am `FillRectangle`-Beispiel (Folie 78ff) aus der Vorlesung.

V4 Bedingungen I

Betrachten Sie folgenden Codeausschnitt:

```

1 Robot karel = new Robot(2,4,North,1);
2 karel.move();
3 if(karel.nextToABeeper()){
4   karel.pickBeeper();
5 }
6 else{
7   karel.putBeeper();
8 }
```

Beschreiben Sie in eigenen Worten, welchem Zweck dieser Codeausschnitt dient. Erweitern Sie außerdem den Code so, dass `karel` nur einen Beeper ablegt, wenn er auch mindestens einen in seiner BeeperBag hat.

V5 Variablen

Legen Sie eine Variable `int a` an und setzen Sie ihren Wert auf 127. Jetzt legen Sie eine weitere Variable `int b` an und setzen Ihren Wert auf 42. Was gibt nun der Ausdruck `int c = a % b` wieder? Beschreiben Sie in Ihren eigenen Worten, für was der `%` Operator verwendet werden kann.

V6 Bedingungen II

Ihr Kommilitone ist etwas tippfaul und lässt deswegen gerne einmal Klammern weg, um sich Arbeit zu sparen. Er hat in seinem Code eine Variable `int number` angelegt, in der er eine Zahl speichert. Ist diese Zahl kleiner als 0, so möchte er das Vorzeichen der Zahl umdrehen und sie anschließend um 1 erhöhen. Ist die Zahl hingegen größer als 0, so möchte er die Zahl verdoppeln. Dazu schreibt er folgenden Code:

```

1 if(number < 0) number = -number;
2 number = number + 1;
3 else number = number * 2;
```

Was passiert beim Ausführen des Codes?

Nachdem Sie ihren Kommilitonen auf den obigen Fehler hingewiesen haben, überarbeitet er seinen Versuch. Wie sieht es mit folgender Variante aus?

```

1 if(counter > 0) counter = counter * 2;
2 if(counter < 0) counter = -counter;
3 counter = counter + 1;
```

Da müssen Sie wohl selbst ran. Erstellen Sie ein korrektes Codestück, um den Sachverhalt korrekt zu implementieren.

V7 Schleifen I

Schreiben Sie den folgenden Ausdruck mithilfe einer `for`-Schleife:

```

1 Robot karel = new Robot(1,1,North,1);
2 int i = 5;
3 while(i < 28){
4   karel.move();
5   i = i + 1;
6 }
```

V8 Schleifen II

Ihr klammerfauler Kommilitone hat auch diesmal wieder zugeschlagen und versucht den Code aus vorheriger Aufgabe kürzer zu schreiben. Was sagen Sie dazu?

```

1 Robot karel = new Robot(1,1,North,1);
2 int i = 5;
3 while(counter < 28) karel.move(); i = i + 1;
```

V9 Anzahl an Umdrehungen

Legen Sie eine Variable `int numberOfTurns` an und setzen Sie ihren Wert zu Beginn auf 0. Erstellen Sie dann einen neuen Roboter und platzieren Sie ihn an der Stelle (3,15). Er schaut dabei nach Westen und hat keine Beeper in seiner Tasche. Lassen Sie den Roboter nun geradewegs auf die Stelle (3,1) zusteuern und alle Beeper auf seinem Weg aufsammeln. Liegen mehrere Beeper auf einer Stelle, so soll er alle Beeper aufsammeln. Bei jedem Aufsammeln, erhöhen Sie den Wert von `numberOfTurns` um 1. Hat er am Ende die Stelle (3,1) erreicht, soll er sich `numberOfTurns`- mal nach links drehen.

V10 Vorsicht Wand!

Gehen Sie in dieser Aufgabe davon aus, dass Sie einen Roboter `karel` an der Position (1,1) erstellt haben und er nach Osten schaut. An der Position (1, x) befindet sich eine Wand, die Avenueposition x ist allerdings unbekannt. Schreiben Sie ein kleines Programm, mit dem Sie den Roboter bis vor die Wand laufen lassen, direkt vor der Wand einen Beeper ablegen, um dann wieder an die Ausgangsposition (1,1) zurückzukehren.

Hinweis: Es gibt die Funktion `frontIsClear()`, mit der getestet werden kann, ob sich in der Blickrichtung des Roboters direkt eine Wand befindet.

V11 Codeverständnis

★ ★ ☆

Beschreiben Sie ausführlich, welches Verhalten der nachfolgende Code umsetzt. Bei Fragen zur Funktionalität einzelner Methoden, werfen Sie einen Blick in die offizielle Dokumentation: <https://csis.pace.edu/bergin/KarelJava2ed/KJRdocs/index.html>.

```

1 Robot robot = new Robot(1, 1, East, infinity);
2 int counter = 0;
3 while(robot.avenue() < World.numberOfAvenues()) {
4   robot.move();
5   counter = counter + 1;
6 }
7
8 robot.turnLeft();
9
10 for(int i = 0; i < counter; i++) {
11   if(i % 2 == 0 && robot.anyBeepersInBeeperBag()) {
12     robot.putBeeper();
13   }
14   robot.move();
15 }
16
17 robot.turnLeft();
18 while(robot.frontIsClear()) {
19   robot.move();
20 }
21
22 robot.turnLeft();
23 while(!robot.areYouHere(1, 1)) {
24   robot.move();
25 }
26
27 robot.turnOff();

```

V12 Navigator

★ ★ ★

Gegeben seien vier Variablen:

<code>int startStreet</code>	<code>int startAvenue</code>
<code>int destinationStreet</code>	<code>int destinationAvenue</code>

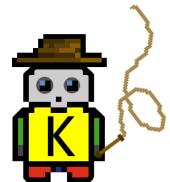
Ihr Roboter befindet sich zu Beginn an der Position (`startStreet`, `startAvenue`) und schaut in eine beliebige Richtung. Schreiben Sie ein Programm, das ihn von dieser Position auf die Position (`destinationStreet`, `destinationAvenue`) laufen lässt.

H Vierte Hausübung

Karel der Schatzsucher

Gesamt 5 Punkte

In dieser vierten Hausübung begleiten Sie den kleinen Roboter **IndianaKarel** auf seiner Schatzsuche. Helfen Sie ihm, indem Sie die drei nachfolgenden Aufgaben bearbeiten! Der Roboter **IndianaKarel** besitzt alle Fähigkeiten eines normalen Roboters und zusätzlich die `turnRight()` Methode. Alle Aufgaben sind voneinander unabhängig, sollten Sie also an einer Stelle Schwierigkeiten haben, stellt dies kein Problem für die anderen Teile dar.



Ihren Lösungscode zu den einzelnen Aufgaben schreiben Sie an die gekennzeichneten Stellen in der Klasse `IndianaKarel.java`. Um die Aufgaben auszuführen, müssen Sie die Klasse `IndianaWorld.java` ausführen und ganz oben die Variable `int exerciseNumber` auf 1,2 oder 3 setzen. **Beachten Sie auch die Hinweise ganz am Ende!**

H1 Pyramide bauen

1 Punkt

In der ersten Aufgabe sollen Sie eine Pyramide aus Beepern zeichnen, in der **IndianaKarel** nach dem Schatz suchen möchte. Zu Beginn befindet sich **IndianaKarel** an Position (3, 1) und blickt nach Osten. Die Anzahl der Beeper steht am Anfang noch auf 0 (Variable `int pyramidBeeper`) und muss von Ihnen durch einen passenden Wert ergänzt werden, sodass am Ende **keine** Beeper mehr in der BeeperBag übrig sind. Zu Beginn zeichnet der Roboter eine Grundlinie der Pyramide aus 13 Beepern in Street 4. Die darauffolgenden Ebenen werden immer mittig platziert und sind genau zwei Avenues kleiner in ihrer Breite als die vorherige Ebene. Außerdem entspricht die Anzahl an Beepern pro Feld immer der aktuellen Streetanzahl - 3. Zur Veranschaulichung betrachten Sie Abbildung 1. Ergänzen Sie den Code so, dass genau diese Pyramide gezeichnet wird. Code zu ergänzen an der Stelle: `public void buildPyramid()`

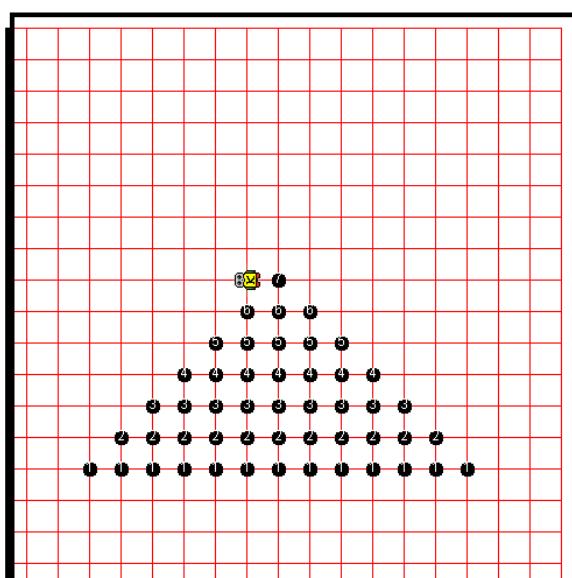


Abbildung 1: Das Bild der fertigen Pyramide

H2 Das Labyrinth**2 Punkte**

Einmal die Pyramide betreten, findet sich `IndianaKarel` in einem großen Labyrinth wieder. Ihre Aufgabe ist es nun ihn aus dem Labyrinth heraus, direkt in die Schatzkammer zu führen. Das Labyrinth besteht aus vielen zufällig generierten Wänden, ein Beispiel finden Sie in Abbildung 2. Der Ausgang ist durch den einzigen Beeper im Labyrinth gekennzeichnet. Dieser soll aufgesammelt werden und `indianaKarel` soll sich danach selbst ausschalten. Der Startpunkt des Roboters ist dabei zufällig in einer der vier Eckpunkte, der Ausgang dann zufällig in einer der drei anderen Eckpunkte. Die Größe der Welt ist ebenfalls variabel und wird zufällig gesetzt. In dieser Aufgabe stehen Ihnen keine Beeper zur Verfügung.

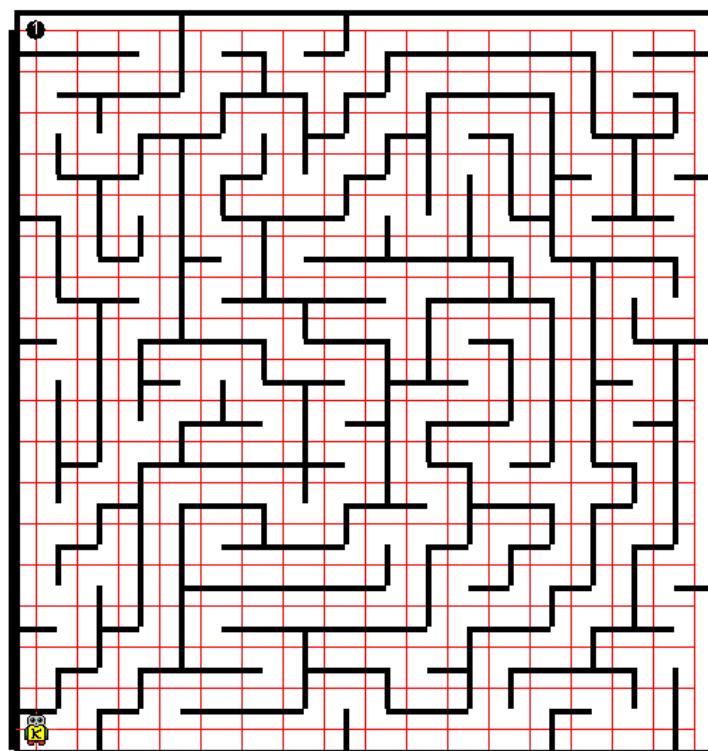


Abbildung 2: Ein zufälliges Beispillabyrinth.

Tipp:

Überlegen Sie sich zunächst einmal, wie Sie selbst durch das Labyrinth gehen würden. Anschließend können Sie diese entwickelte Strategie dann auf `IndianaKarel` ummünzen. Das Labyrinth sieht bei jedem Start anders aus, die einzigen Anhaltspunkte die Sie haben sind die Wände, die immer wieder zufällig erzeugt werden.

Hinweise:

Sie können davon ausgehen, dass keine Säle - also größere Flächen ohne Wände - im Labyrinth vorkommen.

Code zu ergänzen an der Stelle: `public void solveMaze()`

H3 Alles einsammeln**2 Punkte**

Unser kleiner Held ist endlich in der Schatzkammer angekommen und möchte natürlich alle dort liegenden Beeper aufsammeln, die Welt soll also völlig leer danach sein. In dieser Aufgabe, sollen Sie nun eine Strategie für diesen Roboter umsetzen, der ihn alle Beeper in einer Welt einsammeln lässt (ähnlich des PacmanRobot aus der Vorlesung). Die Schwierigkeit dabei: Die Größe der Welt, die Anzahl der Beeper, die Positionen der Beeper und die Startposition des Roboters sind randomisiert (= zufällig gesetzt). Selbstverständlich können auf einem Feld auch mehrere Beeper liegen (siehe auch Abbildung 3). In diesem Fall müssen dann alle Beeper aufgenommen werden. Der Roboter schaut am Anfang immer zufällig in eine der vier Himmelsrichtungen. Code zu ergänzen an der Stelle: `public void collectAll()`

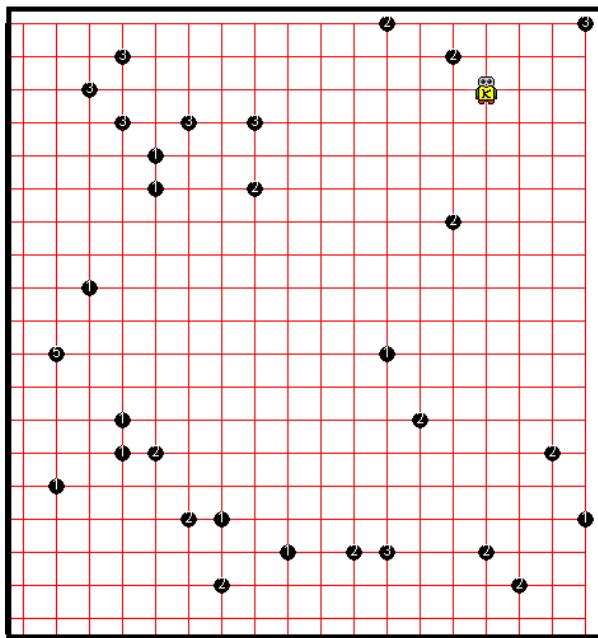


Abbildung 3: Eine zufällige Beispielwelt, in der alle Beeper eingesammelt werden sollen.

Hinweise zur Bearbeitung:

- Beachten Sie unbedingt die Hinweise zur Abgabe auf Übungsblatt 3! Insbesondere sind die Erläuterungen zum korrekten Importieren und Exportieren, sowie zum Plagiarismus wichtig!
- Ergänzen Sie nur den Code an den angegebenen Stellen. Ändern Sie auf keinen Fall etwas an anderen Stellen ab.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.6

Übungsblatt 5

Themen: Referenzsemantik, Klassen, Strings, Arrays

Relevante Folien: KarelJ, Lex. Bestandteile, OO Abstraktion

Abgabe der Hausübung: 30.11.2018 bis 23:55 Uhr

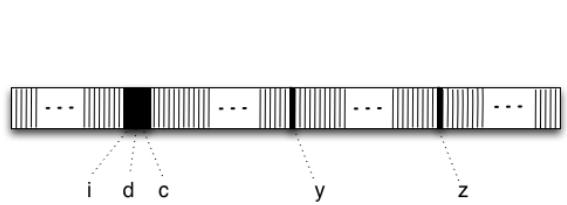
V Vorbereitende Übungen

V1 Referenzen

☆ ☆ ☆

Geben Sie in eigenen Worten wieder, was man unter einer Referenz versteht.

Betrachten Sie außerdem folgendes Schaubild und den Codeausschnitt aus der Vorlesung:



```
1 public class X{  
2     int i;  
3     double d;  
4     char c;  
5     ...  
6 }
```

Zeichnen Sie die Referenzpfeile nach den folgenden Aufrufen ein (ergänzen Sie auch die neuen Reservierungen des Speicherplatzes, wenn nötig):

```
1 X y = new X();  
2 X z = y;  
3 y = new X();
```

V2 Zuweisen und Kopieren

☆ ☆ ☆

Erläutern Sie in Ihren eigenen Worten den Unterschied zwischen Zuweisen und Kopieren. In welchen Fällen sind beide Aktionen synonym zu betrachten?

Wie können Sie eine Zuweisung beziehungsweise eine Kopie in Java umsetzen? Nennen Sie jeweils ein Beispiel.

V3 Arrays

Welche Aussagen zu einem gegebenen Array `a` sind wahr?

- (1) Alle Einträge des Arrays müssen vom selben Typ sein.
- (2) Ein Array hat keine feste Größe und kann beliebig viele neue Einträge hinzufügen.
- (3) Um die Anfangsadresse einer Komponente an Index i zu bekommen, wird i -mal die Größe einer Komponente auf die Anfangsadresse von `a` addiert.
- (4) Außer den eigentlichen Komponenten des Arrays enthält das Arrayobjekt nichts weiteres.
- (5) Ein Array kann nur primitive Datentypen wie zum Beispiel `int`, `char` oder `double` speichern. Somit ist insbesondere nicht möglich, Roboterobjekte in einem Array zu speichern.

Schreiben Sie die nötigen Codezeilen, um ein Array `a` der Größe 42 vom Typ `int` anzulegen. Füllen Sie danach das Array mithilfe einer Schleife, sodass an der Stelle `a[i]` der Wert $2i+1$ steht. Nutzen Sie dabei zuerst eine normale, danach eine verkürzte Schleife.

V4 Wettrennen

Sie haben einen schnellen Roboter `rabbit` erstellt und wollen ihm nun noch ein langsames Gegenstück `turtle` bauen. Beide starten an einem gemeinsamen Punkt, schauen in die gleiche Richtung und besitzen die gleiche Anzahl an Beepern. Sie wollen nun schauen, wer in 10 Runden mehr Strecke zurücklegen kann. Jeder der beiden Kontrahenten kommt pro Runde genau einen Schritt voran, der schnelle `rabbit` erhält jedoch jede zweite Runde sogar einen Extraschritt. Betrachten Sie den folgenden Codeausschnitt, der die Situation implementieren möchte:

```

1 Robot rabbit = new Robot(1,1,East,0);
2 Robot turtle = rabbit;
3
4 for(int i = 0; i < 10; i++){
5
6   if(i / 2 == 0){
7     rabbit.move();
8   }
9
10  rabbit.move();
11  turtle.move();
12
13 }
```

Führen Sie den Code einmal selbst aus und schauen Sie was passiert! Beheben Sie danach **alle** vorhandenen Fehler in der Implementierung, um die oben beschriebene Situation exakt umzusetzen.

V5 Beeper im Laufen ablegen

★ ☆ ☆

In dieser Aufgabe sollen Sie eine erste eigene Methode implementieren. Als Orientierung betrachten Sie die Methode `public void move(int number0fSteps)` der Klasse `FastRobot` aus der Vorlesung. Schreiben Sie nun eine neue Funktion

```
public void beeperMove(int number0fSteps)
```

für den `SymmTurner`-Roboter, den Sie in der Vorlesung kennengelernt haben. Diese soll `number0fSteps` Schritte nach vorne gehen und dabei jedes mal einen Beeper ablegen. Sollte die geforderte Anzahl an Schritten größer sein als die Anzahl an Beepern soll der Roboter einfach stehen bleiben und sich ausschalten (dafür gibt es bereits die Methode `public void turnOff()`). Sollten mehr Beeper vorhanden sein als geforderte Schritte, soll er an seiner finalen Position alle verbleibenden Beeper ablegen.

V6 Himmelsrichtungsdreher

★ ☆ ☆

In vielen Aufgaben reichen uns die eingeschränkten Methoden eines Roboters der KarelJ-Werke nicht. Daher definieren wir uns neue Roboter, welche die technischen Anforderungen erfüllen. Im Foliensatz zu KarelJ haben Sie bereits Beispiele wie den `SymmTurner` Roboter dazu gesehen. In dieser Aufgabe sollen Sie eine neue Roboterklasse definieren, welche sich in alle beliebigen Himmelsrichtungen drehen kann. Dafür ist folgendes Grundgerüst gegeben:

```
1 public class DirectionTurner extends Robot{
2
3     public DirectionTurner (int street, int avenue,
4                             Direction direction, int beepers){
5         super(street, avenue, direction, beepers);
6     }
7
8     // .....
9
10}
```

An der Stelle schreiben wir die neuen Funktionalitäten des Roboters mithilfe von Methoden (vergleiche vorherige Aufgabe). Ergänzen Sie vier neue Methoden namens `public void turnNorth()` etc., damit sich der Roboter gezielt in alle vier Himmelsrichtungen drehen kann.

V7 BeeperMover

★ ☆ ☆

Legen Sie eine neue Roboterklasse `BeeperMover` an. Die `move()`-Methode überladen Sie mit den Anweisungen, die in V5 die Methode `beeperMove` definiert haben. Legen Sie zusätzlich eine neue Funktion `putAllBeepers()` an, welche alle restlichen Beeper aus der Bag ablegt. Nutzen Sie diese Methode, um die `move()`-Methode zu implementieren.

V8 Test auf Gleichheit

★ ★ ☆

Schreiben Sie eine Methode `int strEqual(String a, String b)`. Diese bekommt zwei Strings übergeben und soll bei gleicher Objektidentität der beiden Strings 2, bei Wertegleichheit 1 und bei Wertungleichheit 0 zurückgeben.

V9 Karel, bist du da?

★ ★ ☆

Schreiben Sie eine Methode `boolean karelAreYouThere(String input)`. Diese bekommt einen String übergeben und soll testen, ob sich ein Substring darin befindet, der mit "K" beginnt, mit "rel" endet und dazwischen genau ein Zeichen hat, so wie "Karel". Z.B. soll für die Eingaben "HalloKarel" und "34hfK7relase" `true` zurückgegeben werden.

V10 Ziffern addieren

★ ★ ☆

Schreiben Sie eine Methode `int sumAllDigits(String input)`. Diese soll die Summe aller vorkommenden Ziffern 0-9 im String zusammenaddieren. Sind keine Ziffern im String vorhanden, so soll 0 zurückgegeben werden.

Beispiel: Der Aufruf `sumAllDigits("1DAS0Macht23Spa66ss9")` soll 27 zurückgeben.

Hinweise: Mittels `Character.isDigit(char c)` testen Sie, ob ein gegebener `char` eine Ziffer von 0-9 ist. Mittels `Integer.parseInt(String s)` können Sie einen gegebenen `String`, der aus einer Zahl besteht, zu `int` konvertieren.

V11 TeamRobot

★ ★ ☆

In dieser Aufgabe sollen Sie ihre erste eigene Roboterklasse von Grund auf implementieren. Erstellen Sie dazu eine neue Klasse `TeamRobot`, die die Klasse `Robot` erweitert, also von ihr erbt. Der Konstruktor der Klasse `TeamRobot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und besitzt zusätzlich die Parameter `int left` und `int right`. Der Parameter `int left` gibt an, wie viele zusätzliche Roboter beim Aufruf des Konstruktors links (also westlich) neben des `TeamRobot`s platziert werden. Der Parameter `int right` ist analog, für die Roboter rechts (also östlich). Der `TeamRobot`, sowie die Roboter links und rechts von ihm bilden ein Team. Die zusätzlichen Roboter werden vom `TeamRobot` im Konstruktor erzeugt. Bekommt der `TeamRobot` einen Befehl, so soll dieser von allen Robotern im Team ausgeführt werden. Die zusätzlichen Roboter selbst sind dabei nicht ansprechbar, dass heißt auf ihnen können keine Methoden aufgerufen werden. Überlegen Sie sich, wie Sie die Roboter des Teams in der `TeamRobot`-Klasse speichern können und wie Sie die Befehle die ein `TeamRobot` erhält, an alle Roboter im Team weiterreichen können. Die Befehle meinen hier die Methoden: `move()`, `turnLeft()`, `pickBeeper()` und `putBeeper()`.

Beispiel: Beim Erstellen eines `TeamRobots` mit den Parametern `right = 1` und `left = 2` an der Position (4,4), werden zusätzlich 3 Roboter erstellt, die dem Team angehören, nämlich an den Position (4,2), (4,3) (links) und (4,5) (rechts).

V12 Kopieren



Gegeben seien folgende zwei Klassen:

```

1  public class A {
2      public int number;
3      public String text;
4
5      public A(int nr, String txt) {
6          number = nr;
7          text = txt;
8      }
9
10     public static A copy(A a) {
11         // ...
12     }
13 }
14
15 public class B {
16     public String str;
17     public A a;
18
19     public B(String s, A x) {
20         str = s;
21         a = x;
22     }
23
24     public static B copy(B b) {
25         // ...
26     }
27 }
```

Vervollständigen Sie zuerst die Methode `public static A copy(A a)` der Klasse A. Diese soll eine wertgleiche Kopie des übergebenen Objekts `a` erstellen und zurückgeben. Demnach dürfen die selben Attribute der Kopie und des ursprünglichen Objekts `a` nicht auf dasselbe Objekt verweisen.

Nun sollen Sie ebenfalls die Methode `public static B copy(B b)` der Klasse B vervollständigen. Diese soll eine wertgleiche Kopie des übergebenen Objekts `b` erstellen und zurückgeben.

Um zu testen, ob unsere Implementation der beiden Kopiermethode erfolgreich war, erweitern wir nun die Klasse B um eine Methode `public static boolean isCopy(B original, B copy)`. Dieses soll nun prüfen, ob nicht doch die selben Attribute der Kopie `copy` und des ursprünglichen Objekts `original` auf dasselbe Objekt verweist, ist dies der Fall oder sind die selben Attribute nicht wertgleich, soll `false` zurückgegeben werden. Welche Attribute der Klasse A und B müssen Sie überprüfen? Welche Methode und/oder welcher Vergleichsoperator muss dafür verwendet werden? In welchem Fall müssen Sie nur den Vergleichsoperator verwenden und warum?

H Fünfte Hausübung *RepairBot*

Gesamt 11 Punkte

Auch ein Roboterleben ist hart. Denn selbst wenn unsere mechanischen Freunde nicht von Krankheit betroffen sind, so gibt es auch in Ihrer Welt eine große Gefahr – Verschleiß. Durch das ganze Herumlaufen, Beeper ablegen und Herumdrehen werden die Bauteile unserer Roboter aus den Karel-Werken ziemlich schnell abgenutzt, und so hat sich die Firmenleitung etwas tolles überlegt: In Zukunft sollen sogenannte Repairbots eingesetzt werden, um unsere kaputten Roboter wieder auf Vordermann zu bringen.

Da Sie unsere besten Angestellten in den Karel-Werken sind, ist es Ihre Aufgabe in dieser fünften Hausübung das Ganze sinnvoll umzusetzen!

Nutzen Sie dazu die von uns bereitgestellte Vorlage und halten Sie sich genau an die Vorgaben in den einzelnen Aufgabenteilen.

Don't panic! Der Aufgabentext mag auf den Blick extrem lang erscheinen, die eigentlichen Aufgaben sind dann aber im Vergleich sehr kurz. Lassen Sie sich davon nicht abschrecken, dies ist eine durchaus übliche Situation in der Informatik.

H1 Klasse Battery

1 Punkt

Solange unsere Karel-Roboter noch keine ausgereiften Perpetuum mobile sind, sind auch Sie leider auf einen Akku angewiesen. In diesem Aufgabenteil wollen wir eben diesen umsetzen.

Ganz allgemein sprechen wir bei den Komponenten der Roboter von **Parts**. Diese sind bereits in einer eigenen Klasse umgesetzt und besitzen zwei Strings **name** und **condition**, welche den Namen des Bauteils und seinen aktuellen Zustand beschreiben.

Erstellen Sie eine neue Klasse **Battery** im Package **Parts**, die die Klasse **Part** erweitert, also von ihr erbt.

Die Klasse besitzt ein zusätzliches private-Attribut **int level**. Dieses wird benutzt, um den aktuellen Akkustand der Batterie zu speichern.

Der Konstruktor der Klasse **Battery** hat zwei Parameter **String condition** und **int level**. Wird dieser aufgerufen, soll zunächst der Superkonstruktor mit den Parametern **"Battery"** und **condition** aufgerufen werden und anschließend der Wert des Attributs **level** auf den Wert des übergebenen Parameters **level** gesetzt werden.

Implementieren Sie die Methode **public int getLevel()**, die den aktuellen Akkustand der Batterie zurückgibt.

Implementieren Sie die Methode **public void setLevel(int level)**, diese setzt den aktuellen Akkustand der Batterie auf den Wert des übergebenen Parameters. Sollte der übergebene Parameter kleiner als 100 sein, so ist zusätzlich der Zustand der Batterie auf **Part.conditionUsed** zu setzen. Ist der übergebene Parameter kleiner oder gleich 0, so wird der Zustand der Batterie auf **Part.conditionDamaged** gesetzt.

H2 Klasse Bot**5 Punkte**

Als nächstes folgen unsere normalen und kurzlebigen Roboter, die wir erstellen wollen. Diese Roboter nennen wir Bots, und sie bestehen aus verschiedenen Komponenten, also Parts.

Alle Teilaufgaben in Aufgabe H2 werden in der Klasse Bot im Package `Robots` umgesetzt.

H2.1 Das Grundgerüst der Bot-Klasse**1 Punkt**

Erstellen Sie eine neue Klasse Bot, die die Klasse Robot erweitert, also von ihr erbt. Die Klasse besitzt zwei `private`-Attribute. Ein Array `parts` vom Typ `Part` und ein `boolean waitingForRepair`.

Der Konstruktor der Klasse Bot übernimmt die Parameter des Konstruktors der Oberklasse Robot und soll zunächst den Superkonstruktor aufrufen, im Anschluss soll `waitingForRepair` auf `false` gesetzt werden und das Array `parts` folgendermaßen initialisiert werden:

- (1) Länge des Arrays = 4
- (2) Index 0 des Arrays bekommt ein neues Objekt der Klasse Battery zugewiesen. Der Zustand der Batterie ist `Part.conditionNew`, der Akkustand beträgt 100.
- (3) Index 1 des Arrays bekommt ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "`Camera`", der Zustand ist ebenfalls `Part.conditionNew`.
- (4) Index 2 des Arrays bekommt ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "`Legs`", der Zustand ist `Part.conditionNew`.
- (5) Index 3 des Arrays bekommt ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "`Arms`", der Zustand ist `Part.conditionNew`.

Zum Abschluss dieser Teilaufgabe implementieren Sie noch drei Methoden, um die Bauenteile eines Roboters zu setzen oder zurück zu liefern.

Implementieren Sie die Methode `public Part getPart(int index)`. Diese bekommt einen Index übergeben und gibt das Objekt zurück, dass sich im Array `parts` am übergebenen Index befindet.

Implementieren Sie die Methode `public void setPart(int index, Part part)`. Diese bekommt einen Index sowie ein Objekt der Klasse Part übergeben und weist dem Array `parts` das übergebene Objekt am übergebenen Index zu.

Implementieren Sie die Methode `public int getPartIndexByName(String name)`. Diese bekommt den Namen eines Roboterbauteils als `String` übergeben und gibt den Index des ersten Roboterbauteils im Array `parts` zurück, dessen Name **wertgleich** mit dem übergebenen Name ist. Ist kein Roboterbauteil mit dem übergebenen Namen im Array `parts` vorhanden, soll -1 zurückgegeben werden.

Verbindliche Anforderung: Sie können nicht davon ausgehen, dass sich an Index 0 des Arrays `parts` die Batterie befindet, an Index 1 die Kamera usw.

H2.2 Lauf Bot lauf!**1 Punkt**

Implementieren Sie die Methode `public void faceDirection(Direction dir)`. Diese bekommt eine Himmelsrichtung übergeben und lässt den Roboter in diese blicken. Sollte er bereits in die übergebene Himmelsrichtung blicken, soll nichts passieren (vergleichen Sie dazu auch Aufgabe V6).

Implementieren Sie die Methode `public void randomMove()`. Diese lässt den Roboter zunächst in eine zufällig gewählt Himmelsrichtungen blicken. Nutzen Sie die Klassenmethode `int getRandomNumber(int min, int max)` der mitgelieferten Klasse `MainController`, um eine Zufallszahl zwischen den übergebenen Parametern `min` und `max` zu generieren (beide inklusive). Alle 4 Himmelsrichtungen sollen mit gleicher Wahrscheinlichkeit auftreten. Blickt der Roboter aufgrund der zufällig gesetzten Himmelsrichtung gegen eine Wand, so ist die Methode `turnLeft()` solange aufzurufen, bis dies nicht mehr der Fall ist. Anschließend soll der Roboter einen Schritt in Richtung momentan blickender Himmelsrichtung gehen.

H2.3 Autsch! Der Verschleiß tut weh**1 Punkt**

Implementieren Sie die Methode `public Part checkForDamagedParts()`. Diese gibt das erste Roboterbauteil im Array `parts` zurück, dessen Zustand `Part.conditionDamaged` ist. Ist kein Roboterbauteil beschädigt, soll `null` zurückgegeben werden.

Implementieren Sie die Methode `public void wearOutParts()`. Diese benutzen wir im weiteren Verlauf, um die Abnutzung der einzelnen Roboterbauteile zu simulieren. Später werden wir diese Methode nach jedem gegangenen Schritt des Roboters aufrufen. Nach dem Aufruf der Methode, soll der Akkustand der Batterie des Roboters um 1 reduziert werden. Die Zustände aller Teile im Array `parts` sind entweder `Part.conditionUsed` oder `Part.conditionDamaged`. Ob ein Roboterbauteil nach dem Aufruf als beschädigt deklariert ist, wird zufällig nach folgenden Wahrscheinlichkeiten entschieden:

- (1) Roboterbauteile mit dem Namen "**Camera**" haben eine Wahrscheinlichkeit von 10%, dass Sie durch einen gegangenen Schritt beschädigt werden.
- (2) Roboterbauteile mit dem Namen "**Legs**" haben eine Wahrscheinlichkeit von 22%, dass Sie durch einen gegangenen Schritt beschädigt werden.
- (3) Roboterbauteile mit dem Namen "**Arms**" haben eine Wahrscheinlichkeit von 12.5%, dass Sie durch einen gegangenen Schritt beschädigt werden.

Hierbei ist zu beachten, dass pro Aufruf der Methode `wearOutParts()`, der Zustand von nicht mehr als einem Roboterbauteil auf `Part.conditionDamaged` gesetzt wird (die Batterie ist hierbei auch zu betrachten). Nutzen Sie auch hier wieder die Klassenmethode `int getRandomNumber(int min, int max)` der mitgelieferten Klasse `MainController` um eine Zufallszahl zwischen den übergebenen Parametern `min` und `max` zu generieren (beide inklusive).

Verbindliche Anforderungen: Sie können nicht davon ausgehen, dass sich an Index 0 des Arrays `parts` die Batterie befindet, an Index 1 die Kamera usw. Sie dürfen lediglich davon ausgehen, dass das Array `parts` keine mehrfachen Roboterbauteile mit dem wertgleichen Namen enthält.

Es darf nur die oben genannte Methode `getRandomNumber` zur Generierung von Zufallszahlen benutzt werden, andernfalls führt dies zu Punktabzug!

H2.4 Schwirrt aus, kleine Roboter!

2 Punkte

Zum Abschluss der Klasse `Bot` implementieren Sie die Methode `public void doMove()`. Diese bestimmt welche Aktion ein Roboter der Klasse `Bot` in der aktuellen Runde ausführt. In jeder Runde ruft der `MainController` (Klasse, welche bereits von uns gegeben ist) die `doMove`-Methode eines jeden in der Welt befindlichen Roboters auf. Die `doMove`-Methode soll die folgenden Fälle behandeln:

1. Wartet der Roboter nicht auf eine Reparatur und sind alle Roboterbauteile intakt (Zustand: `Part.conditionUsed` oder `Part.conditionNew`), so geht der Roboter einen Schritt in eine zufällige Richtung und die Methode zur Simulation der Abnutzung der Roboterbauteile wird aufgerufen.
2. Wartet der Roboter nicht auf eine Reparatur und ist ein Roboter teil defekt (Zustand: `Part.conditionDamaged`), so wird der Roboter als wartend deklariert und eine neue `RepairInstruction` mit dem Roboter und dem defekten Roboterbauteil erstellt um diese dann an den `MainController` zu übergeben. Nutzen Sie dazu die Klassenmethode `void orderRepairInstruction(RepairInstruction rep)` der Klasse `MainController`.
3. Wartet der Roboter noch auf eine Reparatur, wobei alle Roboterbauteile intakt sind, so wird der Roboter als nicht mehr wartend deklariert und die Schritte des obigen Falles (Fall 1.) werden ausgeführt.
4. Wartet der Roboter noch auf eine Reparatur und ist ein Roboterbauteil defekt, so ist nichts zu tun.

Zur Erinnerung: Um anzuzeigen, ob ein Roboter auf eine Reparatur wartet, gibt es das Attribut `waitingForRepair`.

H3 Klasse RepairBot**5 Punkte**

Kommen wir nun zu unseren neuen, tollen Reparaturroboter, welche unseren normalen Robotern das Leben erleichtern sollen.

Alle Teilaufgaben in Aufgabe H3 werden in der Klasse RepairBot im Package Robots umgesetzt.

H3.1 Das Grundgerüst der Klasse RepairBot**1 Punkt**

Erstellen Sie eine neue Klasse RepairBot, die die Klasse Bot erweitert, also von ihr erbt. Die Klasse besitzt zwei **private**-Attribute. Ein Array **spareParts** vom Typ Part und ein Attribut **currentJob** vom Typ RepairInstruction. Der Konstruktor der Klasse RepairBot übernimmt die Parameter des Konstruktors der Oberklasse Bot und soll zunächst den Superkonstruktor aufrufen, anschließend soll das Array **spareParts** folgendermaßen initialisiert werden:

- (1) Länge des Arrays = 20
- (2) Indizes 0-4 des Arrays bekommen jeweils ein neues Objekt der Klasse Battery zugewiesen. Der Zustand der Batterie ist **Part.conditionNew**, der Akkustand beträgt 100.
- (3) Indizes 5-9 des Arrays bekommen jeweils ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "**Camera**", der Zustand ist ebenfalls **Part.conditionNew**.
- (4) Indizes 10-14 des Arrays bekommen jeweils ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "**Legs**", der Zustand ist ebenfalls **Part.conditionNew**.
- (5) Indizes 15-19 des Arrays bekommen jeweils ein neues Objekt der Klasse Part zugewiesen. Der Name des Roboterbauteils lautet "**Arms**", der Zustand ist ebenfalls **Part.conditionNew**.

Implementieren Sie die Methode **public int sparePartAvailable(String partName)**. Diese bekommt einen Namen eines Roboterbauteils als String übergeben und gibt den ersten Index des Roboterbauteils im Array **spareParts** zurück, dessen Name **wertgleich** mit dem des übergebenen Namens ist und dessen Zustand entweder **Part.conditionNew** oder **Part.conditionUsed** ist. Sollte kein Roboterbauteil gefunden werden, dass diesen Anforderungen entspricht, so soll -1 zurückgegeben werden.

H3.2 Einmal Service bitte!**2 Punkte**

Implementieren Sie die Methode **public void replacePart(Bot r, int sparePartIndex)**. Diese bekommt einen Roboter der Klasse Bot sowie einen Index des Arrays **spareParts** übergeben. Die Methode tauscht das erste Item im Array **parts** des übergebenen Roboters, dessen Name **wertgleich** ist mit dem Name des Item an Index **sparePartIndex** im Array **spareParts**, mit dem Item an Index **sparePartIndex** im Array **spareParts**.

Implementieren Sie die Methode **public void getClosestToRobot(Robot r)**. Diese bekommt eine Roboter vom Typ Robot übergeben und lässt den RepairBot einen Schritt

näher zum übergebenen Roboter laufen. Befinden sich beide Roboter auf der gleichen Position, so soll nichts passieren.

Hinweis: Der `RepairBot` soll zunächst die Street des zu reparierenden Roboters erreichen, im Anschluss dann die Avenue.

H3.3 Schirrt aus, kleine Reparaturroboter!

2 Punkte

Zum Abschluss der Klasse `RepairBot`, implementieren Sie noch zusätzlich die Methode `public void doMove()`. Diese bestimmt welch Aktion ein Roboter der Klasse `RepairBot` in der aktuellen Runde ausführt. In jeder Runde ruft der `MainController` die `doMove`-Methode eines jeden in der Welt befindlichen Roboters auf. Die `doMove`-Methode soll die folgenden Fälle behandeln:

1. Ist dem Roboter keine aktuelle `RepairInstruction` zugewiesen (`currentJob == null`), so ist zu überprüfen ob die Klassenmethode `getNextRepairInstruction()` der Klasse `MainController` eine neue `RepairInstruction`, die vom Roboter auszuführen ist, zurückgibt oder nicht.
 - (a) Gibt die Methode `getNextRepairInstruction()` `null` zurück, so geht der Roboter einen Schritt in eine zufällige Richtung.
 - (b) Gibt die Methode `getNextRepairInstruction()` eine neue `RepairInstruction` zurück, so wird diese der Variable `currentJob` des Roboters zugewiesen und die unten genannten Schritte zum Ausführen der `RepairInstruction` werden noch in dieser Runde befolgt (Fall 2.).
2. Ist dem Roboter hingegen eine aktuelle `RepairInstruction` zugewiesen, so sind folgende Fälle zu betrachten:
 - (a) Besitzt der Roboter das passende Ersatzteil das zur Ausführung der geordneten `RepairInstruction` nötig ist, so nähert er sich dem zu reparierenden Roboter in jeder Runde um einen Schritt. In der Runde an dem sich beide Roboter auf der gleichen Position befinden, tauscht der `RepairBot` das defekte Roboterbauteil, des liegengeliebenen Roboters, mit dem noch intakten Ersatzteil aus. Sobald die Roboterbauteile getauscht wurden, wird die aktuell zugewiesene `RepairInstruction` verworfen.
 - (b) Besitzt der Roboter kein passendes Ersatzteil, zum Ausführen der geordneten `RepairInstruction`, so wird die aktuell zugewiesene `RepairInstruction` direkt verworfen und der Roboter geht einen Schritt in eine zufällige Richtung.

Beachten Sie den folgenden *Hinweis* für alle obigen vier Fälle:

Rufen Sie **nicht** die Methode zur Simulation der Abnutzung der Roboterbauteile auf. Kein Bauteil eines Roboters der Klasse `RepairBot` nutzt sich in dieser Simulation ab.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.2

Übungsblatt 6

Themen: Klassen, Interfaces, Schleifen, Arrays, Methoden

Relevante Folien: Objektorientierte Abstraktion, Methoden

Abgabe der Hausübung: 07.12.2018 bis 23:55 Uhr

Wichtig und verpflichtend! Dokumentation in Java

Sie erinnern sich sicher an die Dokumentation im Racket-Teil der Veranstaltung. Dort wurde der Typ, sowie eine Beschreibung der Rückgabe über jeder Funktionsdefinition als Kommentar angegeben.

```
;; Type: number number -> number
;; Returns: Euclidean norm of the vector (x,y)
(define (euclid2 x y)
  (sqrt (+ (* x x) (* y y))))
```

Mittels Javadoc und den Tags `@param` und `@return` wollen wir nun auch im Java-Teil eine geeignete Dokumentation unserer Methoden vornehmen. Das Javadoc können Sie automatisch vor jeder Methode mittels `/**` gefolgt von der Enter-Taste generieren. Das Racket-Beispiel oben könnten wir dann folgendermaßen in Java übersetzen:

```
/**
 * @param x first component of two-dimensional vector (x,y)
 * @param y second component of two-dimensional vector (x,y)
 * @return Euclidean norm of the vector (x,y)
 */
double euclid2(float x, float y){
    return Math.sqrt(x*x + y*y); }
```

Genauso wie in den Teilen mit Racket, ist die Dokumentation mit Vertrag Pflicht für alle Javamethoden! Orientieren Sie sich dabei an obigem Beispiel.

Auf Tests wie Sie sie in Racket mit `check-expect` und `check-within` geschrieben haben, können Sie im Java-Teil zunächst verzichten. Die Möglichkeit Ihre Methoden zu testen lernen wir später kennen und werden an geeigneter Stelle nochmals darauf hinweisen, wenn Tests von Ihnen erwartet werden.

V Vorbereitende Übungen

V1 Grundbegriffe



Erklären Sie kurz in eigenen Worten die Unterschiede der folgenden Konzepte zueinander:

1. Klasse vs. Objekt
2. Objekt- vs. Klassenmethoden
3. Abstrakte Klassen vs. Interfaces
4. Überladen von Methoden vs. Überschreiben von Methoden

V2 Lambda I: Java → Racket



In der Vorlesung haben Sie das folgende Interface kennengelernt:

```
public interface IntToDoubleFunction{
    double applyAsDouble(int n); }
```

Sie haben ebenfalls ein Beispiel mittels Lambda-Ausdrücken in Java gesehen:

```
IntToDoubleFunction fct2 = x -> x * 10;
double z = fct2.applyAsDouble(11);
```

Setzen Sie dieses Beispiel nun mittels Lambda-Ausdrücken in Racket um. Die Methode `applyAsDouble` soll ebenfalls implementiert werden.

V3 Lambda II: Racket → Java



Betrachten Sie folgendes Beispiel in Racket:

```
(define (foo x) (lambda (c) (> (* x x) c)))
```

Setzen Sie dieses Beispiel mittels FunctionalInterface und Lambda-Ausdruck in Java um.

V4 Die Würfel sind gefallen!



Mit der Funktion `Math.random()` können Sie eine Zufallszahl im Bereich 0 (inklusive) und 1 (exklusive) erzeugen. Schreiben Sie nun eine Methode `void diceRoll()`.

Diese soll einen Würfelwurf simulieren und die gewürfelte Augenzahl auf der Konsole zurückgeben. Dabei soll der Würfel fair sein, dass heißt alle Augenzahlen sollen mit identischer Wahrscheinlichkeit auftreten.

Hinweise: Überlegen Sie sich, wie Sie die erzeugten Zahlen aus dem Intervall [0, 1) auf die diskrete Menge {1, 2, 3, 4, 5, 6} abbilden können. Mit der Funktion `Math.ceil()` können Sie zur nächstgrößeren, ganzen Zahl aufrunden.

V5 Schleifen

★ ☆ ☆

```

1 int x = 0;
2 while(Math.pow(x,2) <= 1000){
3     x++;
4     System.out.println(x);

```

- (1) Welche Funktion erfüllt der oben stehende Code?
- (2) Ersetzen Sie die `while`-Schleife einmal durch eine `for`- und einmal durch eine `do...while`-Schleife.

V6 Brumm, Brumm, Brumm

★ ☆ ☆

Schreiben Sie eine Klasse `Car` zur Repräsentation von Autos, die folgende Anforderungen erfüllen soll:

- Ein Auto hat einen Namen vom Typ `String` und einen Kilometerstand (`mileage`) vom Typ `double`. Beide Attribute sollen `private`, nicht `public`, sein.
- Der Konstruktor soll einen String als Parameter erhalten, der den Namen des Autos angibt. Der Konstruktor soll den Namen des Autos setzen und den Kilometerstand auf 0.0 setzen.
- Schreiben Sie die Methoden `public double getMileage()` und `public String getName()`. Diese liefern die entsprechenden Attribute der Klasse `Car` zurück.
- Schreiben Sie die Methode `public void drive(double distance)`, die eine Distanz in Kilometern als Argument erhält und auf den alten Kilometerstand addiert.

V7 Gleicher Abstand

★ ★ ☆

Schreiben Sie eine Methode `static boolean evenlySpaced(int a, int b, int c)`, welche genau dann `true` zurückliefert, wenn der Abstand zwischen dem kleinsten und dem mittleren Element genauso groß ist wie der Abstand zwischen dem mittleren und dem größten Element. Dabei kann jeder der Parameter `a`, `b` oder `c` das kleinste, mittlere oder größte Element sein. Die Klasse, zu der die Methode gehört, muss nicht implementiert werden.

V8 Aufsummieren

★ ★ ☆

Schreiben Sie eine Methode `void sumUp(int[] a)`, die ein Array `a` von Typ `int` erhält. An Index $i \in \{0, \dots, a.length-i\}$ in `a` soll nun der neue Wert $a[0] + \dots + a[i]$ geschrieben werden. Dabei bezeichnen $a[0], \dots, a[i]$ die Werte in `a` unmittelbar vor dem Aufruf der Methode.

(Beispiel auf nächster Seite)

Übergeben wir der Funktion das folgende Array `a = [3, 4, 1, 9, -5, 4]`, so wird das Array folgendermaßen modifiziert:

→ [3, 3+4, 3+4+1, 3+4+1+9, 3+4+1+9+(-5), 3+4+1+9+(-5)+4]

→ [3, 7, 8, 17, 12, 16]

V9 Spieglein, Spieglein...



Wir nennen eine Gruppe von Elementen in einem Array Spiegel, wenn sie irgendwo im Array nochmal auftaucht, nur in umgekehrter Reihenfolge. Beispielsweise ist im Array [7,6,5,1,9,8,5,6,7] ein Spiegel vorhanden und zwar [7,6,5]. Schreiben Sie eine Methode `int maxMirror(int[] arr)`. Diese bekommt ein Array übergeben und gibt die Länge des größten Spiegels im übergebenen Array zurück. Gibt es keinen Spiegel so wird einfach 0 zurückgeliefert.

Hinweis: Starten Sie mit zwei Zeigern auf dem ersten und dem letzten Element. Vergleichen Sie nun paarweise die Elemente und überlegen Sie sich, wann Sie die beiden Zeiger weiter in die Mitte bewegen.

V10 Matrix-Multiplikation Reloaded



Erinnern Sie sich an die zweite Hausübung? Dort haben Sie die Matrixmultiplikation bereits in Racket implementiert. Wir wollen dies auch in Java umsetzen, und verwenden statt verschachtelten Listen nun zweidimensionale Arrays.

Der folgende Code stellt beispielsweise die Matrix $\begin{pmatrix} 5 & 8 \\ 1 & -3 \end{pmatrix}$ dar.

```
int [][] matrix = new int [2] [2];
matrix [0] [0] = 5;
matrix [0] [1] = 8;
matrix [1] [0] = 1;
matrix [1] [1] = -3;

// alternativ und kuerzer: int [][] matrix = {{5,8},{1,-3}}
```

Sie sehen also, dass Sie einem Array in Java beliebig viele Dimensionen geben können.

Schreiben Sie eine Methode `int [][] matrixMul(int[][] mat1, int[][] mat2)`.

Die Methode bekommt zwei Matrizen, dargestellt durch zwei zwei-dimensionale Arrays, übergeben und gibt die resultierende Produktmatrix zurück. Sollte die Multiplikation aufgrund falscher Dimensionen nicht möglich sein, so geben Sie eine entsprechende Nachricht auf dem Bildschirm aus und liefern `null` zurück.

Hinweis: Verwenden Sie drei ineinander geschachtelte `for`-Schleifen. Die erste iteriert über die Reihen von `mat1`, die zweite iteriert über die Spalten von `mat1` und die Reihen von `mat2` und die letzte iteriert über die Spalten von `mat2`.

V11 Statischer und dynamischer Typ



```

1  public class Alpha {
2    protected int v;
3    public Alpha(int a) {
4      v = a;  }
5
6    public class Beta extends Alpha {
7      public Beta(int b, int c) {
8        super(b);
9        v = c;  }
10
11   public Alpha x1() {
12     super.v++;
13     return new Beta(0, v);  }
14
15   public int x2(int x) {
16     return x + ++v + v++;  }
17
18  public class Gamma extends Beta {
19    private short y;
20    public Gamma(int d, int e) {
21      super(d, e);
22      y = (short) d;  }
23
24    public int x2(int x) {
25      return x - y;  }
26
27  public static void main(String[] args) {
28    Alpha a = new Alpha(7);
29    Beta b = new Beta(0, 1);
30    Gamma g = new Gamma(9, 2);
31    a = b.x1();
32    int t = b.x2(5);
33    a = new Beta(10, 12).x1();
34    b = g;
35    int r = g.x2(50);  }
  }
```

Hinweis: Nach Zeile X heißt unmittelbar nach X, noch vor Zeile X+1.

- (1) Welchen statischen und dynamischen Typ haben **a**, **b** und **g** nach Zeile 30?
- (2) Welchen statischen und dynamischen Typ hat **a** und welchen Wert hat **a.v** nach Zeile 31?
- (3) Welchen Wert haben **t** und **b.v** nach Zeile 32?
- (4) Welchen statischen und dynamischen Typ haben **a**, **b** und welchen Wert hat **a.v** nach Zeile 34?
- (5) Welchen Wert haben **r** und **b.v** nach Zeile 35?

V12 The final Countdown



Schauen Sie folgenden Java-Code an. Beheben Sie sämtliche eingebauten Fehler, um den Code lauffähig zu machen. Was müssen Sie im Code ändern, um die folgende Ausgabe zu erhalten?

"Die Abschlussklausur von FOP ist am 9.4.2019"

```
1 public final class A {  
2  
3     private int value1 = 0, value2 = 0;  
4     private final int value3 = 0;  
5  
6     private int getValue1() {  
7         return value1; }  
8  
9     private int getValue2() {  
10        return value2; }  
11  
12    private void setValue1(int newValue1) {  
13        value1 = newValue1; }  
14  
15    private void setValue2(int newValue2) {  
16        value2 = newValue2; }  
17  
18    public final void changeValue3(final int newValue3) {  
19        value3 = 0; }  
20}  
21  
22 class B extends A {  
23  
24     public void changeValue3(final int newValue3) {  
25         value3 = newValue3; }  
26  
27     public static void main(String args[]) {  
28         B obj = new B();  
29  
30         obj.setValue1(9);  
31         obj.setValue2(4);  
32         obj.value3 = 2019;  
33  
34         String result = "Die Abschlussklausur von FOP ist am "  
35         + getValue1() + "." + getValue2() + "." + obj.value3  
36  
37         System.out.println(result);  
38     }  
39 }
```

V13 Klassen, Interfaces und Methoden

★ ★ ★

V13.1

Schreiben Sie ein `public`-Interface A mit einer Objektmethode `m1`, die Rückgabetyp `double`, einen `int`-Parameter `n` und einen `char`-Parameter `c` hat.

V13.2

Schreiben Sie ein `public`-Interface B, das von A erbt und zusätzlich eine Objektmethode `m2` hat, die keine Parameter hat und einen `String` zurückliefert.

V13.3

Schreiben Sie eine `public`-Klasse XY, die A implementiert, aber `m1` nicht. Klasse XY soll ein `protected`-Attribut `p` vom Typ `long` haben sowie einen `public`-Konstruktor mit Parameter `q` vom Typ `long`. Der Konstruktor soll `p` auf den Wert von `q` setzen. Weiter soll XY eine `public`-Objektmethode `m3` mit Rückgabetyp `void` und Parameter `xy` vom Typ XY haben, aber nicht implementieren.

V13.4

Schreiben Sie eine public-Klasse YZ, die von XY erbt und B implementiert. Die Methode `m1` soll `n+c+p` zurückliefern und `m2` den `String "Hallo"`. `m3` soll den Wert `p` von `xy` auf den Wert `p` des eigenen Objektes addieren. Der Konstruktor von YZ ist `public`, hat einen `long`-Parameter `r` und ruft damit den Konstruktor von XY auf.

V14 Jedes dritte Element

★ ★ ★

Gegeben sei eine Klasse X. Schreiben Sie für diese Klasse die `public`-Objektmethode `foo`. Diese hat ein Array `a` von Typ `int` als formalen Parameter und liefert ein anderes Array `b` vom Typ `int` zurück, das aus `a` entsteht, indem jedes dritte Element gelöscht wird, das heißt, die Elemente von `a` an den Indizes $0, 3, 6, 9, \dots$ werden nicht nach `b` kopiert, alle anderen Elemente von `a` werden in derselben Reihenfolge, wie sie in `a` stehen, nach `b` kopiert. Weitere Elemente hat `b` nicht. Sie dürfen voraussetzen, dass `a` mindestens Länge 2 hat und ungleich `null` ist. Sie dürfen einfach Operator `=` für das Kopieren von Elementen verwenden.

Hinweis: Überlegen Sie sich die Gesetzmäßigkeit, nach der die Indizes $1, 2, 4, 5, 7, 8, \dots$ in `a` auf die Indizes $0, 1, 2, 3, 4, 5, \dots$ in `b` abzubilden sind. Für die Länge von `b` werden Sie eine Fallunterscheidung benötigen, je nachdem, welchen Rest `a.length` dividiert durch 3 ergibt. Denken Sie auch an die letzten beiden Elemente von `a`.

H Sechste Hausübung *Approximationen*

Gesamt 10 Punkte

Auf diesem Übungsblatt haben Sie verschiedene Konzepte neu kennengelernt oder vertieft. Dazu gehören Zufallszahlen, mehrdimensionale Arrays, verschachtelte Schleifen, Klassen und Interfaces. In dieser sechsten Hausübung wollen wir diese Werkzeuge nutzen, um uns dem Thema Approximation - also Näherungsverfahren - zu widmen.

Die folgenden drei Aufgaben sind allesamt unabhängig voneinander. Vergessen Sie nicht, Ihre Methoden gemäß den am Anfang des Blattes vorgestellten Konventionen zu dokumentieren!

H1 Approximation von π

2 Punkte

In der ersten Aufgabe schauen wir uns zwei verschiedene Wege zur Approximation der Kreiszahl π an. Dazu finden Sie in der Vorlage die Klasse `ApproximatePi`, in der Sie zwei Methoden zu vervollständigen haben. Beim Ausführen der Klasse `Main` wird automatisch ein Fehlerdiagramm erzeugt, welches die Güte unserer Annäherung aus den beiden Teilaufgaben visualisiert.

H1.1 Monte-Carlo-Simulation

1 Punkt

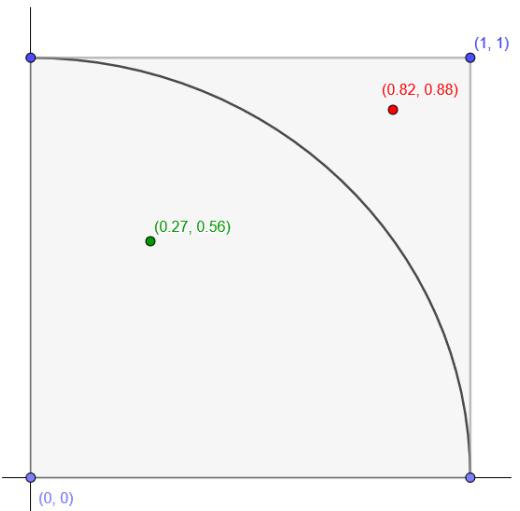
Monte-Carlo-Simulationen gehören in die Klasse der randomisierten Algorithmen und können zur Flächeninhaltsberechnung verwendet werden. Dieses Verfahren wollen wir hier anhand der probabilistischen Berechnung der Kreiszahl π verdeutlichen.

Wir stellen uns zunächst ein Quadrat mit der Seitenlänge 1 vor. Zusätzlich betrachten wir einen Viertelkreis mit Radius 1 innerhalb dieses Quadrates. Wir erzeugen nun zufällig einen Punkt in diesem Quadrat und überprüfen, ob der Punkt innerhalb oder außerhalb des Kreises liegt.

Machen wir dies mit vielen Punkten, so gilt der Zusammenhang:

$$\frac{\pi}{4} = \frac{r^2 \cdot \pi}{(2 \cdot r)^2} \approx \frac{\text{Anzahl Punkte innerhalb des Kreises}}{\text{Anzahl aller Punkte}}$$

Betrachten Sie die Abbildung rechts für eine Darstellung des Sachverhaltes. Der zufällig erzeugte grüne Punkt ist ein Treffer innerhalb des Kreises, der rote nicht.



Vervollständigen Sie die Methode `double monteCarloPi(int n)`. Diese bekommt die Anzahl der zu generierenden Punkte `n` übergeben und liefert eine Approximation für π basierend auf dem oben beschriebenen Verfahren zurück.

H1.2 Flächeninhaltsberechnung

1 Punkt

Der exakte Flächeninhalt des Kreises kann auch über ein Integral bestimmt werden:

$$\int_0^1 \frac{4}{1+x^2} dx$$

Da wir dieses Integral nicht so einfach analytisch in Java lösen können, greifen wir auch hier auf eine Approximation zurück. Dafür verwenden wir hier die sogenannte numerische Trapezintegration.¹ Zur Approximation eines Integrals kann dann die folgende Formel verwendet werden:

$$\int_a^b f(x) dx \approx \frac{h}{2} \sum_{k=1}^n (f(x_{k-1}) + f(x_k)) \quad \text{mit } h = \frac{b-a}{n} \quad \text{und } x_k = a + k \cdot h$$

Vervollständigen Sie die Methode `double integrationPi(int n)`. Diese bekommt die Schrittgröße `n` (ein größeres `n` steht für eine feinere Unterteilung und damit eine bessere Approximation) übergeben und gibt die Approximation des obigen Integrales zurück.

In Abbildung 1 sehen Sie die Visualisierung der Trapezintegration. Sie erstellen Trapeze unter der Kurve von f (hier grün) und berechnen deren Flächeninhalt zur Approximation.

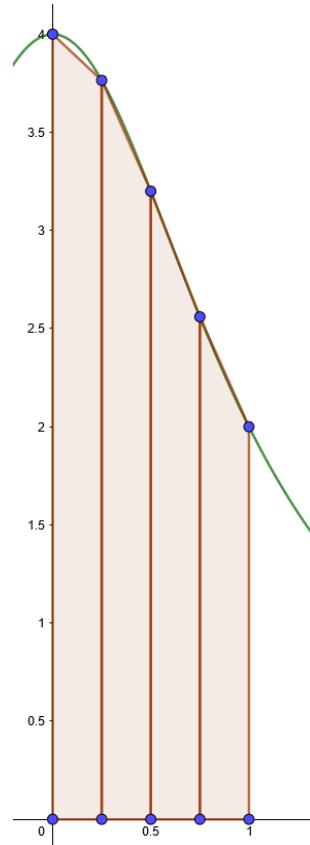


Abbildung 1: Beispiel zur Trapezintegration mit $h = 0.25$

¹https://en.wikipedia.org/wiki/Trapezoidal_rule

H2 Bayer Pattern

3 Punkte

Jede Kamera gibt heutzutage RGB Bilder aus. Trotzdem verwenden die meisten von ihnen keine drei separaten Chips zur Farbspeicherung, sondern lediglich einen Chip, der auf einem sogenannten *Color Filter Array* basiert. An jeder Stelle im Bild wird also zunächst nur einer der drei RGB-Werte gespeichert, und anschließend werden die beiden fehlenden Werte interpoliert. Um dies zu realisieren, kann das sogenannte *Bayer Pattern* verwendet werden. Nach diesem Pattern werden die Werte folgendermaßen gespeichert: Abbildung 2 zeigt, wie die Farbwerte des Bildes abgespeichert werden und Abbildung 3 zeigt ein beispielhaftes Bild, welches in diesem Muster gespeichert wurde:

rGb	Rgb	rGb	Rgb	rGb
rgB	rGb	rgB	rGb	rgB
rGb	Rgb	rGb	Rgb	rGb
rgB	rGb	rgB	rGb	rgB
rGb	Rgb	rGb	Rgb	rGb



Abbildung 2: Ausschnitt des Patterns: Der jeweils gespeicherte Farbwert ist groß.

Abbildung 3: Ein Bild gespeichert im Bayer Pattern (entnommen hier).

Um aus diesem Pixel-Muster nun wieder ein normales Bild herzustellen, werden die fehlenden Werte für jeden Pixel interpoliert. Dies bedeutet Sie berechnen für einen grünen Pixel den R- und den B-Wert, für einen roten Pixel den B- und G-Wert und für einen blauen Pixel den R- und G-Wert aus den jeweiligen Werten der Nachbarschaft. Wir betrachten in dieser Aufgabe die einfachste aller Interpolationen - das arithmetische Mittel. Um also an der Stelle eines grünen Pixels den fehlenden Rotwert zu bestimmen, nehmen Sie alle direkten roten Nachbarpixel (egal ob diagonal, vertikal oder horizontal) und bilden den Mittelwert dieser Werte. Die zwei fehlenden Werte werden also immer durch den Mittelwert der Nachbarwerte geschätzt. Wenn Sie dann für jeden Pixel die zwei fehlenden Werte nach diesem Schema berechnet haben, erhalten Sie das rekonstruierte Bild. Für das Beispielbild würde damit beispielsweise Abbildung 5 entstehen:



Abbildung 4: Originaler Ausschnitt.



Abbildung 5: Rekonstruierter Ausschnitt.

H2.1 Einführung

0 Punkte

Machen Sie sich in dieser Teilaufgabe kurz mit den bereits implementierten Klassen `Image` und `BayerPattern` der Vorlage vertraut.

Die Klasse `Image` ermöglicht es Ihnen PNG-Bilder einzulesen und wieder abzuspeichern. Wir verwenden, wie auch in Hausübung 02, den RGB-Farbraum. Jedes Pixel nimmt dabei genau die Farbe an, die durch sein sogenanntes RGB-Tripel beschrieben wird. Sollten Sie hier nochmal eine Auffrischung Ihres Wissens benötigen, schauen Sie im entsprechenden Text auf Übungsblatt 02 nach.

Um die Pixel eines Bildes zu speichern, verwenden wir ein dreidimensionales `int`-Array. Der erste Index gibt an, an welcher Höhe sich ein Pixel im Bild befindet, der zweite Index gibt an, an welcher Breite sich ein Pixel im Bild befindet und der dritte Index gibt an, welcher von den drei Farbwerten (rot, grün oder blau) dort gespeichert wird. Dabei gilt für die dritten Indizes folgende Zuordnung:

$$[] [] [0] \rightarrow \text{rot}$$

$$[] [] [1] \rightarrow \text{grün}$$

$$[] [] [2] \rightarrow \text{blau}$$

Somit erhält man beispielsweise den Blauwert des Pixels an der oberen linken Ecke über diese Indizes: `[0] [0] [2]`.

Zum Einlesen von Bildern können Sie die bereits vorhandenen Konstruktoren `Image(String filePath)` und `Image(int[][][] data)` verwenden. Der erste Konstruktor bekommt den Pfad zu einer PNG-Datei übergeben und speichert dieses Bild im Objektattribut `data` ab. Der zweite Konstruktor bekommt ein Bild als dreidimensionales `int`-Array übergeben und weist dem Objektattribut `data` das übergebene Array zu.

Die ebenfalls implementierte Methode `void saveAsPNG(String filePath)` können Sie nutzen, um ein Bild am übergebenen Dateipfad abzuspeichern.

Die Klasse `BayerPattern` ermöglicht es Ihnen über den Konstruktor `BayerPattern(String filePath)` eine Datei im Bayer Pattern zu laden. Das Bayer Pattern wird als zweidimensionals `int`-Array im Objektattribut `data` gespeichert. Auch hier beschreiben die Indizes, wie auch in der Klasse `Image`, die Position eines Pixels bzw. eines Farbwerts im Bayer Pattern. Welcher der drei Farbwerte für einen Pixel gespeichert wird, ist durch den Aufbau des Bayer Patterns (Abbildung 2) gegeben.

Die oben beschriebenen Methoden müssen Sie in den beiden Teilaufgaben verwenden, um am Ende Ihren fertigen Code auszuprobieren. In der Vorlage werden Ihnen bereits Beispieldateien mitgeliefert. Die Methoden, die Sie zu schreiben haben, befinden sich alle in der Klasse `BayerPattern` und müssen nur noch von Ihnen ergänzt werden.

H2.2 Drei getrennte Farbarrays**1 Punkt**

Ergänzen Sie zu Beginn in der Klasse `BayerPattern` die Objektmethode

```
public int[][][] splitColorChannels().
```

Diese benutzt das Array `int[][] data` der Klasse `BayerPattern`, welche das Bayer Pattern repräsentiert (siehe Abbildung 2). Die Methode soll nun alle Farbwerte trennen und in verschiedenen Ebenen eines neuen dreidimensionalen Arrays speichern. Ist das übergebene Array $h \times w$ groß, so ist das zurückgegebene Array $h \times w \times 3$ groß. Die letzte "Dimension" des Arrays soll dabei angeben, welcher Farbwert dort gespeichert wird. Die Beschreibung welcher Farbwert an welche Stelle geschrieben werden soll, finden Sie in obiger Teilaufgabe H2.1.

Die fehlenden Farbwerte an den jeweiligen Positionen setzen Sie auf -1.

Unverbindlicher Hinweis:

Überlegen Sie sich, welche Regelmäßigkeit Sie finden können, in welchen Mustern die Farben im Bayer Pattern auftreten. Wie können Sie beispielsweise schnell überprüfen, welche Farbe sich im Bayer Pattern an der Stelle (x, y) befindet?

H2.3 Interpolation der fehlenden Farbwerte**2 Punkte**

Ergänzen Sie nun in der Klasse `BayerPattern` die `public`-Klassenmethode

```
int[][][] interpolateMissingValues(int[][][] splittedColorChannels).
```

Diese bekommt ein Array übergeben, das über die Methode `int[][][] splitColorChannels()` erzeugt wurde und soll nun mittels Interpolation alle fehlenden Farbwerte berechnen (siehe Anleitung in H2) und das daraus resultierende Array zurückgeben. Im zurückgegebenen Array darf nun keine -1 als Farbwert auftreten!

Achtung mit den Pixeln am Rand! Diese haben eine andere Nachbarschaft als die innen liegenden Pixel und müssen separat von Ihnen betrachtet werden. Hier wird das arithmetische Mittel dann von der geringeren Anzahl an Nachbarn berechnet, es werden keine Pixel außerhalb des Arrays dazu gedacht oder Ähnliches.

Sollte durch die Bildung der Mittelwerte keine natürlichen Zahlen mehr auftreten, runden Sie zur **nächstkleineren Zahl** ab!

Unverbindliche Hinweise:

- Nutzen Sie die Regelmäßigkeiten im Muster, die Sie bereits in Aufgabe H2.2 erkannt und verwendet haben.
- Suchen Sie auch hier wieder eine Regelmäßigkeit, und zwar welche Nachbarschaftspixel Sie in Ihre Berechnung einbeziehen müssen. Wie können Sie schnell an alle Nachbarschaftspixel für eine gegebene Postion (x, y) kommen?

H3 Polynome**5 Punkte**

Die Approximation durch Polynome stellt in der Analysis ein mächtiges Werkzeug dar, und so wollen wir uns in dieser Aufgabe damit beschäftigen, Polynome in Java zu modellieren. Ein Polynom n -ten Grades hat die folgende funktionelle Form:

$$p(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \quad n \geq 0$$

Dabei nennen wir die Werte a_0, \dots, a_n die Koeffizienten des Polynoms. Als Grad des Polynoms bezeichnen wir den höchsten Exponenten n , für den der Koeffizient a_n im Ausdruck $a_n x^n$ nicht 0 ist (auch Leitkoeffizient genannt). Ein Polynom $n - ten$ Grades besitzt also $n+1$ Koeffizienten, wobei die Koeffizienten mit Ausnahme des Leitkoeffizienten auch den Wert 0 annehmen können.

Weiter ist in dieser Aufgabe nichts vorgegeben, Sie werden alles von Grund auf selbst modellieren und implementieren.

H3.1 Interface für Funktionen**1 Punkt**

Vielleicht wollen wir später auch noch andere Funktionstypen als Polynome betrachten. Daher legen wir uns zunächst ein Interface für Funktionsfamilien an. Schreiben Sie ein **public**-Interface **Function** mit einer Objektmethode **getValue**, die Rückgabetyp **double** und einen **double** Parameter hat.

H3.2 Generelle Polynome**1 Punkt**

Neben den Polynomen aus der elementaren Algebra, welche wir hier betrachten, finden Polynome auch Anwendungen in vielen anderen Bereichen. Schreiben Sie eine **public**-Klasse **GeneralPolynomial**, die **Function** implementiert, aber **getValue** nicht.

Klasse **GeneralPolynomial** soll ein **private**-Attribut **coefficients** vom Typ **double[]**, ein **private**-Attribut **degree** vom Typ **int** sowie einen **public**-Konstruktor mit Parameter **coef** vom Typ **double[]** haben. Der Konstruktor soll das Attribut **coefficients** auf **coef** setzen und **degree** auf den Grad des Polynoms. Dabei stellt das Array **coefficients** die Koeffizienten des Polynoms dar. Die Koeffizienten sind dabei absteigend angeordnet, für fehlende Koeffizienten wird 0 eingetragen.

Ein Beispiel:

$$s(x) = \underbrace{4x^3 - 5x^2 - 7}_{\text{Polynom 3. Grades}} \rightarrow \underbrace{[4, -5, 0, -7]}_{\text{a.length = 4}}$$

Weiter soll die Klasse **GeneralPolynomial** die beiden **public**-get-Methoden **double[] getCoefficients()** und **int getDegree()** besitzen, welche die entsprechenden Attribute zurückliefern. Zusätzlich besitzt Sie die **private**-set-Methoden **void setCoefficients(double[] coefficients)** und **void setDegree(int degree)**, welche die entsprechenden Attribute setzt.

Weiter besitzt die Klasse die **public**-Objektmethoden (beide sind beide vom Rückgabetyp **GeneralPolynomial**) **firstDeriv()** und **antiDeriv()**, implementiert diese aber nicht.

H3.3 Auswertung, Ableitungen und Stammfunktionen

3 Punkte

Nun kommen wir zu den für uns interessanten Polynomen. Schreiben Sie eine public-Klasse **Polynomial**, die von **GeneralPolynomial** erbt und **Function** implementiert.

Der Konstruktor von **Polynomial** ist **public**, hat einen **double[]**-Parameter **coef** und ruft damit den Konstruktor von **GeneralPolynomial** auf.

Methode **getValue** soll das Polynom an der übergebenen Stelle **x** auswerten und die Auswertung zurückgeben.

Methode **firstDeriv** soll die erste Ableitung des Polynoms bilden und als neues Polynom zurückgeben. Die Länge des neuen Koeffizientenarrays ist dabei immer um genau 1 kleiner! Das bedeutet, sollte der vorletzte Koeffizient 0 sein, so wird im neuen Array eine 0 an letzter Stelle stehen.

Methode **antiDeriv** soll die Stammfunktion des Polynoms bilden und als neues Polynom zurückgeben. Dabei ist die Länge des neuen Arrays stehts um 1 größer und enthält an der letzten Stelle den Koeffizienten 0.

Beispiel für das Polynom $s(x) = 3x^3 + 0.333x^2 + 5$:

Der Code...

```

1 double[] coeffs = {3,0.333,0,5};
2 Polynomial s = new Polynomial(coeffs);
3 Polynomial sD = s.firstDeriv();
4 Polynomial sAD = s.antiDeriv();
5 System.out.println(Arrays.toString(s.getCoefficients()));
6 System.out.println(Arrays.toString(sD.getCoefficients()));
7 System.out.println(Arrays.toString(sAD.getCoefficients()));
8 System.out.println(s.getValue(4));

```

... liefert folgende Konsolenausgabe (inklusive Kommentare von uns):

```

1 [3.0, 0.333, 0.0, 5.0] // Koeffizienten
2 [9.0, 0.666, 0.0] // erste Ableitung
3 [0.75, 0.111, 0.0, 5.0, 0.0] // Stammfunktion
4 202.328 // Wert an der Stelle x = 4

```

Zur Erinnerung:

Die allgemeine Ableitung eines Polynoms n -ten Grades:

$$p'(x) = a_1 + 2a_2x + 3a_3x^2 + \dots + na_nx^{n-1} = \sum_{i=1}^n ia_i x^{i-1} = \sum_{i=0}^{n-1} (i+1)a_{i+1}x^i$$

Die allgemeine Stammfunktion eines Polynoms n -ten Grades:

$$\int p(x) dx = a_0x + \frac{1}{2}a_1x^2 + \frac{1}{3}a_2x^3 + \dots + \frac{1}{n+1}a_nx^{n+1} = \sum_{i=0}^n \frac{a_i}{i+1}x^{i+1} = \sum_{i=1}^{n+1} \frac{a_{i-1}}{i}x^i$$

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.2

Übungsblatt 7

Themen: Assertions, JUnit und Fehlerbehandlung

Relevante Folien: Fehlerbehandlung

Abgabe der Hausübung: 14.12.2018 bis 23:55 Uhr

Einbinden und Verwendung von JUnit

Auf diesem Übungsblatt werden wir erstmals mit JUnit arbeiten. In dieser Veranstaltung verwenden wir die aktuellste Version, also JUnit 5. JUnit 5 wird bereits mit Eclipse ausgeliefert und muss daher nicht erneut heruntergeladen werden. Allerdings muss die Bibliothek dem Projekt noch hinzugefügt werden. Dazu führt man die folgenden Schritte aus:

1. Öffnen des Eclipse-Projekts, an dem gearbeitet und für das die Bibliothek genutzt werden soll.
2. Öffnen der Projekt-Eigenschaften durch Rechtsklick auf dem Projektnamen.
3. Auswahl des Eintrags *Java Build Path*.
4. Im rechten Teil des Fensters zum Karteireiter *Libraries* wechseln und auf *Add Library...* klicken.
5. In dem nun erscheinenden Dialog wählt man *JUnit*, klickt auf *Next* und wählt dann JUnit 5.
6. Nun kann man auf *Finish* klicken.

V Vorbereitende Übungen

V1 Theoriefragen

★ ★ ★

V1.1 Grundlegendes

1. Wie hängen die Begriffe `throws` und `throw` zusammen? Wo wird was verwendet?
2. Ist es sinnvoll, eigene Exceptionklassen zu definieren? Welche Vorteile ergeben sich hieraus?
3. Nennen Sie die Methoden der Assert-Klasse, die Sie zum Testen bei einem typischen JUnit Testcase in der Vorlesung kennengelernt haben, und beschreiben Sie kurz deren Verwendung.

V1.2 Wahr oder falsch?

Welche der folgenden Aussagen ist wahr?

- (A) Auf einen `try`-Block muss immer mindestens ein `catch`-Block folgen.
- (B) Wenn Sie eine Methode schreiben, die eine Exception auslösen könnte, müssen Sie diesen riskanten Code mit einem `try/catch`-Block umgeben.
- (C) Auf einen `try`-Block können beliebig viele verschiedene `catch`-Blöcke folgen.
- (D) Eine Methode kann nur eine einzige Art von Exception werfen.
- (E) Die Reihenfolge der `catch`-Blöcke ist grundsätzlich gleichgültig.
- (F) Laufzeit (Runtime)-Exceptions müssen behandelt oder deklariert werden.
- (G) Es darf kein Code zwischen `try` und `catch` geschrieben werden.
- (H) Eine Methode wirft eine Exception mit dem Schlüsselwort `throws`.

V2 Try/Catch-Block

★ ★ ★

Was ist das Problem mit dem folgenden Codeausschnitt?

```
public static void main(String[] args) {  
    int [] arr = new int [10];  
    System.out.println(arr[77]);  
}
```

Modifizieren Sie den Code mittels `try/catch`-Blockes um das Problem zu beheben.

V3 Exceptions



Sehen Sie sich den folgenden Code genau an (ExplodeException erbt von Exception).

- (1) Welche Ausgabe wird dieses Programm beim Aufruf der Methode `test()` liefern?
- (2) Welche Ausgabe erfolgt bei einer Änderung von Zeile 4 in String `test = "yes";` ?

```

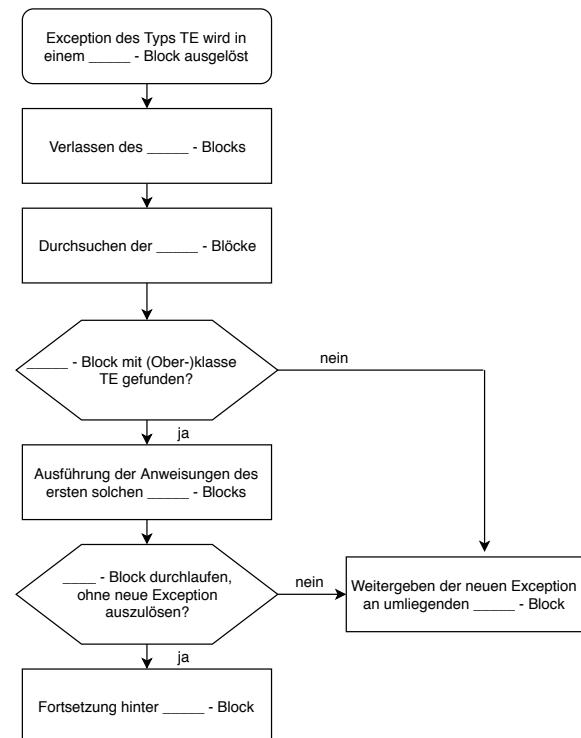
1 public class ExceptionTest {
2
3     public void test() {
4         String test = "no";
5         try {
6             doRisky(test);
7         } catch (ExplodeException ex) {
8             System.out.println("catching ExplodeException!");
9         }
10    }
11    public void doRisky(String test) throws ExplodeException{
12        System.out.println("begin doRisky");
13        if(test.equals("yes")){
14            throw new ExplodeException();
15        }
16        System.out.println("end doRisky");
17        return; } }
```

V4 Ablaufdiagramm



In dieser Aufgabe beschäftigen wir uns mit dem Auffangen einer hypothetischen Exception des Typs TE. Ergänzen Sie in nebenstehenden Ablaufdiagramm an freien Stellen, ob es sich um einen `catch`- oder `try`-Block handelt.

Ein Beispiel für Ablaufdiagramme finden Sie beispielsweise hier:
<https://de.wikipedia.org/wiki/Programmablaufplan#Beispiel>



V5 assert-Anweisungen

★ ☆ ☆

In der Vorlesung haben Sie die assert-Anweisungen kennengelernt.

1. Beschreiben Sie in eigenen Worten, was ein Assertion-Error ist.
2. Schreiben Sie den nachfolgenden Codeschnipsel kompakter mittels assert-Anweisungen!

```
if( (k > 0 && k + 1 <= 5) || (k % 3 == 2) )
    throw new AssertionError("Very bad k!");
```

3. Sie wissen, dass assert-Anweisungen beim Kompilieren an- oder abgeschaltet werden können mit entsprechenden Setzungen für den Compiler. Welche Vorteile ergeben sich hieraus? Warum sollten wir sie ausschalten und nicht einfach auch im realen Einsatz des Programms immer eingeschaltet mitlaufen lassen?

V6 Erster Test mittels BeforeEach

★ ★ ☆

In dieser Aufgabe wollen wir einen Blick auf die `BeforeEach`-Annotation von JUnit5 werfen. Methoden mit dieser Annotation werden **vor Beginn jedes einzelnen Tests** ausgeführt!

Gegeben sei eine Bibliothek für Geometrie-Funktionen. Dabei betrachten wir nur die Funktion `triangleArea`, die den Flächeninhalt eines Dreiecks berechnet und dazu die Längen der einzelnen Seiten (`a`, `b` und `c`) als `int`-Werte übergeben bekommt.

V6.1 Setup vor jedem Test

Gegeben sei folgende Klasse `GeoLib`:

```
1 public class GeoLib{
2
3     public GeoLib(){ }
4
5 }
```

Um die Funktionen der Bibliothek verwenden zu können, muss zunächst ein Objekt vom Typ `GeoLib` erzeugt werden. Hierzu können Sie den parameterlosen Standard-Konstruktor der Klasse `GeoLib` verwenden. Geben Sie eine entsprechend mit JUnit-Annotationen versiegene Methode namens `setup` an, die in einer Testklasse steht und die für jeden Testfall eine neue Instanz von `GeoLib` in dem bereits deklarierten Attribut `geoLib` speichert.

V6.2 Testfall

Geben Sie mindestens drei JUnit-Test an, die überprüfen, ob die Methode `triangleArea` für verschiedene Dreiecke korrekt arbeitet. Mindestens ein Testdreieck sollte dabei auch entartet sein.

V7 Fehlertypen



Zur Erinnerung: Die Syntax ist die Struktur und die Semantik die Bedeutung des Programms. Dies lässt uns zwei verschiedene Arten von Fehlern definieren:

1. **Syntaxfehler** findet und moniert der Compiler. Typische Beispiele sind falsch geschriebene oder unbekannte Worte oder eine falsche Anordnung von Worten und Operanden.
2. Bei **Semantikfehlern** unterscheiden wir nochmals in...
 - ... solche, die der Compiler nicht findet, sondern die sich zur Laufzeit durch Programmabbruch auswirken; und
 - ... solche, die der Compiler findet, wie zum Beispiel nicht erreichbarer Programmcode oder auch Typfehler.

(1) Beheben Sie im folgenden Codeausschnitt alle lexikalischen und syntaktischen Fehler:

```

1 public int[] reverseArray (int[] source) throw Exception {
2     int length = source.length();
3     int[] inverted;
4     int i=0;
5
6     try {
7         inverted = new int[length];
8
9         while (i < length){
10             inverted[i] = source[i];
11             i++;
12         }
13     }
14     catch (Exception) {
15         System.out.println("Caught Exception... ");
16     }
17     catch (IndexOutOfBoundsException e) {
18         System.out.println("Caught Exception: " + e);
19     }
20     inverted = new int[length()];
21     inverted = source ;
22
23     return inverted;
24 }
```

(2) Was passiert generell beim Aufruf der Methode `reverseArray`? Warum kann der Code auch ohne vorhandene syntaktische Fehler nicht kompiliert werden? Was müsste man beheben um den Code kompilierbar zu machen?

(3) Das Programm läuft zwar jetzt fehlerfrei, das Ergebnis entspricht aber noch nicht dem gewünschten Ergebnis (Array soll umgedreht werden). Beheben Sie alle fehlerhaften Stellen im Code, um das gewünschte Ergebnis zu erreichen.

V8 Testen mit JUnit - Qualitätskontrolle



Wir wollen ein neues System zur Qualitätskontrolle in einer Produktionskette testen. Hierzu gibt es eine Klasse `ProductLineManagement`, die Güter (Typ `Product`) herstellt. Ihre Tests sollen nun prüfen, ob dies schnell genug und hinreichend gut erfolgt. Die Maschinen garantieren dabei immer eine Mindestqualität von 89 (=89% des Optimums).

Die Qualität wird gemessen auf einer Skala von 0 (defekt) bis 100 (perfekt).

Die Testklasse deklariert ein Attribut `static ProductLineManagement plm`, auf das Sie zugreifen können.

V8.1 Setup-Methode

Vor jedem Test muss die (sehr komplexe) Produktionskette initialisiert werden. Dies erfolgt durch den Aufruf des Konstruktors der Klasse `ProductLineManagement` mit dem Namen der Firma als String. Den Firmennamen dürfen Sie beliebig wählen. Geben Sie eine mit JUnit-Annotationen versehene öffentlich sichtbare Methode an, die diese Initialisierung vor jedem Test durchführt.

V8.2 Normalfall

Schreiben Sie einen Test für eine normale Produktion. Hierbei soll ein einziger Artikel `normalProduct` durch die Methode `Product produce(String)` der Klasse `ProductLineManagement` (siehe oben) produziert werden. Stellen Sie sicher, dass der gelieferte Artikel nicht `null` ist und eine Mindestqualität - abfragbar via `getQuality()` - von 89 besitzt. Als Titel des Artikels können Sie einen beliebigen String angeben.

V8.3 Behandlung von Exceptions

Schreiben Sie nun einen weiteren Test der auch wie im vorherigen Aufgabenteil ein neues Produkt erstellt. Nur diesmal reichen Sie dieses Produkt mittels `boolean submit(Product, int)` aus der Klasse `ProductLineManagement` für die Qualitätskontrolle ein. Wurde die gewünschte Qualität erreicht, liefert die Methode `true`, andernfalls wirft die Methode eine `InsufficientQualityException`.

Testen Sie das Verhalten und das Auftreten der Exception mit einem Produkt, indem Sie dieses einmal auf die (garantierte) Mindestqualität von 89 und einmal auf die unerreichbare Qualität von 101 testen.

V9 Testen: Racket und Java



Sie haben nun sowohl das Testen in Java mittels JUnit, als auch das Testen in Racket mittels Checks kennengelernt. In dieser Aufgaben sollen Sie zuerst eine Problemstellung in beiden Sprachen lösen und anschließend Ihre Implementierungen testen.

Gegeben ist eine Zahlenliste. In Racket ist diese als Liste von `numbers` gegeben, in Java als Array von Typ `int []`. Außerdem sind zwei Parameter `lower` und `upper` gegeben. Ziel ist es, alle Werte aus der Zahlenliste zu sortieren, welche nicht zwischen diesen beiden Grenzwerten `lower` und `upper` liegen (jeweils exklusive).

Ergänzen Sie die beiden untenstehenden Codeausschnitte und Verträge, um diese Problemstellung zu lösen.

Racket:

```
;; Type: (list of number) number number -> (list of number)
;; Returns:
(define (numbersBetween alon lower upper)
  ....)
```

Java:

```
/**
 *
 * @param arr
 * @param lower
 * @param upper
 * @return
 */
public int[] betweenNumbers(int[] a, int lower, int
    upper){
  ....}
```

Sollte der Parameter `lower` dabei größer als der Parameter `upper` sein, so soll in Java eine `LowerBiggerThanUpperException` geworfen werden. Ergänzen Sie dies in Ihrer Implementierung.

In Racket haben Sie die Möglichkeit einen Fehler auszulösen. Dies geschieht über den Befehl (`error msg`), wobei `msg` ein String mit der gewünschten Fehlermeldung ist. Konventionsmäßig einigen wir uns darauf, dass wir bei `msg` zuerst den Funktionsnamen nennen, gefolgt von einem Doppelpunkt und einer Beschreibung des Fehlers. Lösen Sie äquivalent zur `LowerBiggerThanUpperException` auch in der Racketfunktion einen Fehler für diesen Fall aus.

Testen Sie abschließend die beiden Implementierungen mit jeweils 3 Tests. Ein Test sollte dabei das korrekte Werfen der Fehlermeldung testen. In Racket können Sie dies mit (`check-error fct msg`) bewerkstelligen. Dabei steht (`fct`) für den Funktionsaufruf und `msg` für die Fehlermeldung.

V10 Exceptionklassen

★ ★ ☆

V10.1

Schreiben Sie eine `public`-Klasse `MyException`, die von `Exception` erbt. Der Konstruktor dieser Klasse hat einen Parameter `str` vom Typ `String` und einen Parameter `n` vom Typ `int`. Ein Objekt von `MyException` hat ein `private`-Attribut `message` vom Typ `String`. Der Konstruktor weist `message` die Konkatenation aus beiden Parametern zu. Die `public`-Methode `getMessage` von `Exception` soll so überschrieben werden, dass `message` zurückgeliefert wird.

V10.2

Schreiben Sie eine `public`-Klasse `X` mit einer `public`-Klassenmethode `km`, die einen `int`-Parameter `n` hat, `int` zurückliefert und potentiell `MyException` wirft. Und zwar wirft `km` eine `MyException` mit "`n can not be negative`" und `n` als Parameterwerten, wenn `n` negativ ist. Andernfalls liefert `km` das Quadrat von `n` zurück.

V10.3

Schreiben Sie eine `public`-Klasse `Y` mit einer `public`-Objektmethode `m`, die einen `int`-Parameter `n` hat und `int` zurückliefert. Diese Methode ruft `km` von `X` mit `n` auf, ohne ein Objekt von `X` dafür einzurichten, und liefert das Ergebnis von `km` zurück. Sollte `km` eine Exception werfen, dann soll die Botschaft der Exception auf dem Bildschirm ausgegeben werden.

V11 Welcher Belag darf es sein?

★ ★ ☆

Schreiben Sie eine Klasse `NoBreadException` welche von `Exception` erbt, im Konstruktor einen Parameter `String` `topping` erhält und damit den Konstruktor der Basisklasse mit der Konkatenation "`There is no bread, only` + `topping` aufruft.

Schreiben Sie dann ein Functional Interface namens `Lunch`. Dieses enthält die funktionale Methode `String getTopping(String s)`, welche eine `NoBreadException` wirft.

Initialisieren Sie nun das Functional Interface `Lunch` durch einen Lambda-Ausdruck. Geprüft werden soll, ob der String ein korrektes Sandwich ist. Dabei besteht ein korrektes Sandwich aus zweimal dem Substring "`bread`" und einem Topping dazwischen. Korrekte Sandwichs sind also beispielsweise "`breadtunabread`" oder "`breadabcdefgbread`". Vor dem ersten "`bread`" und nach dem zweiten "`bread`" darf kein Substring mehr stehen. Zurückgegeben werden soll immer das Topping, also der Substring zwischen den beiden "`bread`'s. Das Topping muss dabei nicht „sinnvoll“ sein, sondern irgendein beliebiger String. Ist kein Brot vorhanden, so soll eine `NoBreadException` mit dem alleinigen Topping geworfen werden.

Sie dürfen davon ausgehen, dass niemals nur eine Brotscheibe verwendet wird, sondern entweder zwei oder keine.

V12 Arrays, Exceptions und Vererbung



V12.1 Klasse X

Gegeben sei die folgende Klasse:

```
public class X {  
    public int a[];  
    public boolean writable[];  
}
```

Wir benutzen das Array `a[]` um `int`-Werte zu speichern. Im Array `writable[]`, wird festgehalten ob ein `int`-Wert im Array `a[]` überschrieben werden darf, oder nicht. Der `int`-Wert `a[i]` darf überschrieben werden, wenn `writable[i] == true`. Sie können davon ausgehen, dass beide Arrays der Klasse `X` immer die gleiche Länge besitzen, sodass die Indizes der beiden Arrays übereinstimmen.

Implementieren Sie den Konstruktor der Klasse `X`, dieser bekommt einen `int`-Parameter übergeben und initialisiert die Arrays `a[]` und `writable[]` mit der gleichen Länge, dabei soll jeder Wert im Array `writable[]` mit `true` initialisiert werden. Die Länge beider Arrays entsprechen hier dem Wert des übergebenen Parameters. Erweitern Sie nun die Klasse um eine `public`-Methode `save`. Die Methode liefert nichts zurück, bekommt einen `int`-Parameter übergeben und speichert den übergebenen Parameter am kleinsten freien Index im Array `a[]` ab, an dem ein Wert nach aktueller Definition von `writable` überschrieben werden darf. Zusätzlich setzt sie den entsprechenden Wert im Array `writable[]` auf `false`. Darf kein Wert überschrieben werden, so soll die Methode eine `ArrayStoreException` mit der Nachricht "`no free space left`" werfen.

V12.2 Klasse Y

Schreiben Sie nun eine Klasse `Y`, die von der Klasse `X` aus Aufgabe V12.1 erbt. Die Klasse soll die Methode `save` der Oberklasse überschreiben. Die Methode liefert nichts zurück, bekommt einen `int`-Parameter übergeben und soll mit diesem Parameter die Methode `save` der Oberklasse aufrufen. Wird dabei eine `ArrayStoreException` geworfen, so soll diese abgefangen werden. Ist dies der Fall, so sollen die beiden Arrays `a[]` und `writable[]` um ihre aktuelle Länge erweitert werden, um dann anschließend den übergebenen Parameter mittels der Methode `save` abspeichern zu können. Die bereits gespeicherten Werten in den beiden Arrays dürfen bei der Erweiterung nicht verloren gehen.

H Siebte Hausübung

Terme und Arithmetik

Gesamt 13 Punkte

In der Mathematik ist ein Term ein sinnvoller Ausdruck, der Zahlen, Variablen, Symbole für mathematische Verknüpfungen und Klammern enthalten kann. In dieser siebten Hausübung wollen wir uns mit solchen Termen und grundlegender Arithmetik beschäftigen.

H1 Eigene Exception implementieren

1 Punkt

Schreiben Sie zunächst eine neue Klasse `InvalidTermException` im Package `Exceptions`, die von der Klasse `Exception` abgeleitet ist. Die neue `Exception` besitzt zwei `private String`-Attribute `term` und `message`, sowie die `public`-Methoden `String getTerm()` und `String getMessage()`, die die jeweiligen Attribute zurückgeben. Der Konstruktor der `Exception` bekommt als ersten Parameter den Term übergeben, der die `Exception` ausgelöst hat und als zweiten Parameter eine Botschaft übergeben und weist die übergebenen Werte den entsprechenden Objektattributen zu.

H2 Terme modellieren

3 Punkte

Wir wollen nun zuerst eine Klasse implementieren, die es uns ermöglicht Terme abzuspeichern und im weiteren Verlauf der Aufgabenstellung auszuwerten. Schreiben Sie nun eine Klasse `Term` im Package `Main` mit zwei `private String`-Attributnen `term` und `result`. Der Konstruktor der Klasse bekommt einen angeblichen Term als `String` übergeben und entfernt zunächst alle Whitespaces im übergebenen `String`. Anschließend wird geprüft, ob nun dieser `String` einem sinnvollen Ausdruck entspricht, also ob die Zahlen, Klammern und Symbole für mathematische Verknüpfungen in genau diesem `String` sinnvoll angeordnet sind und ob somit dieser `String` wirklich einen Term darstellt. Der Einfachheit halber, wollen wir in unserer Implementation auf Variablen in Termen verzichten. Terme können somit aus folgenden Elementen bestehen:

(1) **Zahlen:**

Entweder als ganze Zahl oder als Gleitkommazahl mit einem Punkt als Dezimaltrennzeichen. Negative Zahlen tragen ein Minus (-) als Vorzeichen.

Beispiele für Zahlen: -3 oder 5.239.

(2) **Symbole für mathematische Verknüpfungen:**

Wir beschränken uns auf die Grundrechenarten, also auf die vier mathematischen Operationen Addition, Subtraktion, Multiplikation und Division. Dargestellt durch deren Symbole: Plus (+), Minus (-), Mal (*) und Geteilt (/).

(3) **Klammern:**

Zur Gruppierung von Termen benutzen wir nur runde Klammern.

Hinweis: Eine Auslassung des Malzeichens, wie beispielsweise bei dem Term $3(2 + 1)$, ist nicht zulässig! Der korrekte Term lautet dann $3 * (2 + 1)$.

Folgend einige Beispiele für Strings, die zulässige Terme darstellen:

- (1) "120"
- (2) "3*2+4-2"
- (3) "(5+5.25)*2-10/(2+1)"
- (4) "0+3.2*(-4+-5)*((2+2)*(1+(6-2.25125)))"

Folgend einige Beispiele für Strings, die **keine** zulässigen Terme darstellen:

- (1) "3++2" → Der Additionsoperator darf nicht mehrmals aufeinander folgen.
- (2) "(3+2*(2-4)" → Eine schließende Klammer fehlt.
- (3) "4(4+8)" → Ein Operator fehlt.

Sollte der übergebene String kein Term sein, so ist eine `InvalidTermException` mit einer Botschaft zu werfen, die beschreibt, wieso der übergebene String eben gerade kein Term ist, also gegen welche Anforderung verstoßen wurde. Passen Sie den Methodenkopf des Konstruktors der Klasse `Term` dahingehend an!

H3 Binäre Operatoren

1 Punkt

Wir wollen nun eine Klasse implementieren, die es uns ermöglicht die vier Grundrechenarten mit jeweils zwei Operanden vom Typ `String` auszuführen, um damit im weiteren Verlauf der Aufgabe unsere Terme auswerten zu können. Dazu schreiben wir eine Klasse `DoubleStringMath` im Package `Math`, die das Interface `StringMath` implementiert. Alle vier von Ihnen zu implementierenden Methoden, die die Grundrechenarten darstellen, sollen zunächst die beiden übergebenen Operanden zum Typ `double` konvertieren und dann damit die entsprechende Berechnung durchführen. Die Ergebnisse der Berechnungen sollen wieder als `String` zurückgegeben werden. Nutzen Sie dafür die Klassenmethode `String Utils.doubleToString(double d)`, diese konvertiert die übergebene Zahl vom Typ `double` zum Typ `String`.

H4 Terme auswerten

4 Punkte

Erweitern Sie nun die Klasse `Term` um eine Methode `public String getResult()`. Die Methode überprüft zunächst, ob das Objektattribut `result` noch `null` ist. Ist dies der Fall, so wird der Term solange ausgewertet, bis er nur noch aus einer Zahl besteht. Diese Zahl, also das Ergebnis der Auswertung, wird dann `result` zugewiesen und von der Methode zurückgegeben. Ist das Objektattribut `result` ungleich `null`, so wird es direkt zurückgegeben. Um einen Term auszuwerten, wenden Sie die Rechenregeln in folgender Reihenfolge an:

- (1) Klammern
- (2) Punktrechnung (Multiplikation und Division)
- (3) Strichrechnung (Addition und Subtraktion)
- (4) Von links nach rechts

Führen Sie die für die Auswertung benötigten Berechnungen zunächst über ein Objekt der von Ihnen geschriebenen Klasse `DoubleStringMath` durch. Dafür weisen Sie dem Klassenattribut `public static StringMath math` der Klasse `Utils`, zu einem geeigneten Zeitpunkt, ein neues Objekt der Klasse `DoubleStringMath` zu und greifen auf dieses Klassenattribut innerhalb der von Ihnen zu implementierenden Methode `public getResult()` der Klasse `Term` zu, um die Berechnungen durchzuführen. Dies ermöglicht uns einen einfachen Austausch der Klasse, die für die arithmetischen Operationen zuständig ist. Wollen wir die zuständige Klasse austauschen, so weisen wir dem Klassenattribut `math` der Klasse `Utils`, einfach ein Objekt einer anderen Klasse, die das Interface `StringMath` implementiert, zu.

Unverbindliche Hinweise:

Sie können sich folgende Methoden anlegen, um einen Term auszuwerten:

1. `boolean isAtom()`: Gibt `true` genau dann zurück, wenn der Term nur aus einer einzigen Zahl besteht wie beispielsweise unter (1) in den Beispielen von Aufgabe H2 gezeigt.
2. `int findAddOrSubOperation()`: Gibt die Position des linken Operanden einer Addition oder Subtraktion innerhalb eines Strings wieder.
3. `int findMulOrDivOperation()`: Gibt die Position des linken Operanden einer Multiplikation oder Division innerhalb eines Strings wieder.
4. `int findInnerMostExpression()`: Gibt die Position des meist innerliegenden Ausdrucks innerhalb des Terms zurück, also genau den Ausdruck, der nach Prioritäten als erstes berechnet werden muss.
5. `String evaluateSimpleExpression(String simple)`: Bekommt einen einfachen Term als String übergeben, also einen Term ohne Klammern, und liefert dessen Ergebnis als String zurück. Ein Beispiel für solch einen einfachen Term finden Sie in den Beispielen von H2 unter (2).

Folgend ein Beispiel für die Auswertung eines Terms, Schritt für Schritt:

1. `"54 + (545 * 234 + (4545 - 54))* 6 + 34"`
2. `"54 + (545 * 234 + 4491)* 6 + 34"`
3. `"54 + 132021 * 6 + 34"`
4. `"792214"`

H5 TermIO

2 Punkte

Jetzt wollen wir eine Klasse schreiben, die es uns ermöglicht Terme aus Dateien zu laden und die anschließend erfolgende Auswertung der Terme in einer Datei zu speichern. Legen Sie zunächst eine neue Klasse `TermIO` im Package `Main` an. Die Klasse hat nur ein `public`-Array `terms` vom Typ `Term`, in diesem Array wollen wir die Terme speichern, die wir aus einer beliebigen Datei laden wollen.

H5.1 Terme aus Datei einlesen**1 Punkt**

Schreiben Sie eine Methode `public boolean readTermsFromFile(String filePath)`. Die Methode bekommt einen Pfad zu einer angeblichen Datei, die Terme enthalten soll, übergeben und benutzt die Klassenmethode

```
public static String[] readAllLinesFromFile(String filePath) throws
FileNotFoundException, UnsupportedEncodingException
```

der Klasse `Utils`, um die Datei Zeile für Zeile in ein `String`-Array einzulesen. Sofern eine `FileNotFoundException` geworfen wird, fangen Sie diese ab und geben Sie mittels `System.out.println()` die Nachricht `"File not found: "`, konkateniert mit dem übergebenen Dateipfad auf der Konsole aus und geben Sie `false` zurück. Sollte eine `UnsupportedEncodingException` geworfen werden, so geben Sie die Nachricht `"Unsupported encoding: "` konkateniert mit der Botschaft der Exception auf der Konsole aus und geben Sie ebenfalls `false` zurück. Wird keine Exception beim Lesen der Datei geworfen, so sollten sich nun alle Zeilen der Datei im zurückgegebenen `String`-Array der Methode `readAllLinesFromFile` befinden. Das Array `terms` der Klasse `TermIO` wird nun mit der Länge des `String`-Arrays initialisiert und anschließend wird jeder `String` des Arrays benutzt um ein entsprechendes `Term`-Objekt zu erstellen, das dem Array `terms` zugewiesen wird. Mit dem `String` an Index 0 im `String`-Array erstellen Sie ein neues `Term`-Objekt ebenfalls am Index 0 im Array `terms` usw. Sollte beim Erstellen eines neuen `Term`-Objektes, eine `InvalidTermException` geworfen werden, so fangen Sie diese ab und geben die Nachricht `"Invalid term: "` konkateniert mit dem Term auf der Konsole aus und geben sie `false` zurück. Sollten keine `Exceptions` geworfen werden, so gibt die Methoden am Ende `true` zurück.

H5.2 Terme auswerten und in Datei speichern**1 Punkt**

Schreiben Sie die Methode `public boolean writeTermResultsToFile(String filePath)` der Klasse `TermIO`. Diese bekommt einen Pfad zu einer Datei übergeben und nutzt die Methode `public static void writeLinesToFile(String filePath, String[] lines)` `throws IOException` der Klasse `Utils` um die Ergebnisse der Auswertungen der Terme im Array `terms` in einer Datei abzuspeichern. Das Ergebnis der Auswertung jedes Terms im Array `terms` soll jeweils in einer neuen Zeile in der Datei am übergebenen Dateipfad abgespeichert werden. Fangen Sie eine eventuell geworfene `IOException` ab und geben Sie die Nachricht `"Error writing to file: "`, konkateniert mit der Botschaft der `Exception` auf der Konsole aus und geben Sie `false` zurück. Sollten keine `Exceptions` geworfen werden, so gibt die Methoden am Ende `true` zurück.

H6 Testen mittels JUnit**2 Punkte**

Abschließend wollen wir einen Teil unserer implementierten Methoden testen. Legen Sie dazu eine neue Klasse `TermTests` im ebenfalls von Ihnen anzulegenden Package `Tests` an. Implementieren Sie nun folgende Tests in der neu angelegten Klasse:

- (1) 3 Tests, die den Konstruktor der Klasse `Term` mit korrekten Termen aufrufen. Hierbei soll der Aufruf des Konstruktors keine `Exception` werfen.
- (2) 3 Tests, die den Konstruktor der Klasse `Term` mit Strings aufrufen, die keine Terme

darstellen. Diese Tests sollen also prüfen, ob der Konstruktor in diesem Falle eine `InvalidTermException` wirft.

- (3) 3 Tests, die die Methode `getResult()` der Klasse `Term` überprüfen.

Achtung: Selbstverständlich gilt auch hier, genau wie im Racket-Teil der Veranstaltung, dass Ihre Tests auch nicht-triviale Eingaben testen sollen! Von den drei Tests muss mindestens einer davon einen Randfall abdecken!

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v2.4

Übungsblatt 8

Themen:	Generics und Collections
Relevante Folien:	Generics und Collections
Abgabe der Hausübung:	21.12.2018 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Theoriefragen

★ ★ ★

1. Welche Vorteile ergeben sich durch die Nutzung von Generics?
2. Diskutieren Sie Vor- und Nachteile von Collections gegenüber herkömmlichen Arrays unter den folgenden Aspekten:
 - (a) Finden von Elementen
 - (b) Einfügen neuer Elemente
 - (c) Löschen von Elementen
3. Auch in Racket haben Sie Listen kennengelernt. Ist das Java-Interface `java.util.list` vergleichbar mit den Listen aus Racket?

V2 Collections und Exceptions

★ ★ ★

Gegeben sei folgender Codeausschnitt:

```
1 double foo(double[] numbers, double n) {  
2     LinkedList<Double> list = new LinkedList<Double>();  
3     for (double x : numbers) {  
4         if (x > 0 && x <= n && x % 2 != 0) {  
5             list.add(x);  
6         }  
7     }  
8     Collections.sort(list);  
9     return list.getLast();  
10 }
```

- (1) Beschreiben Sie kurz und bündig, aber präzise und unmissverständlich was der oben gegebene Code macht.

(2.1) An welcher Stelle kann im Code eine Exception geworfen werden? Durch welche Eingaben wird sie ausgelöst?

(2.2) Modifizieren Sie den Code mithilfe eines try/catch-Blockes so, dass in diesen Fällen die Nachricht der Exception auf der Konsole ausgegeben wird.

V3 A-well-a bird bird bird, bird is the word Part I ★☆☆

In dieser Aufgabe betrachten wir eine stark reduzierte Typ hierarchie zur Modellierung von Vögeln. Dabei stellen die Pfeile die Erbbeziehungen zwischen Klassen dar. Dazu ist das folgende Typdiagramm in Abbildung 1 gegeben. Hier ist **Bird** also die Oberklasse und die drei anderen Klassen sind Erben dieser.

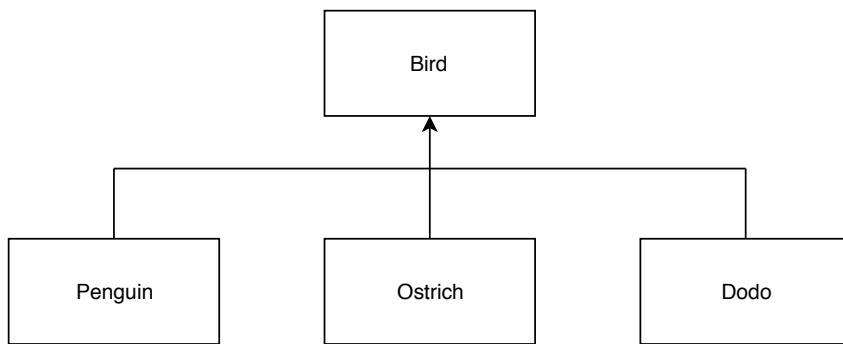


Abbildung 1: Typhierarchie mit drei Vogelarten

Wir nutzen die generische Datenstruktur `Vector<E>`. Dabei beschränken wir uns auf die Methode `void add(E entry)`, die ein Element vom Typ `E` in den Vector einfügt.

(1): Deklarieren und initialisieren Sie eine Variable `v` mit dem **statischen** Basistyp `List` und dem **dynamischen** Typ `Vector`, so dass darin genau Objekte der Typen `Birds`, `Penguin`, `Ostrich` und `Dodo` gespeichert werden können. Nutzen Sie generische Typparameter!

(2): Geben Sie Java-Code an, um in den obigen Vector `v` jeweils ein neues Element vom Typ `Penguin` und `Ostrich` einzufügen. Sie dürfen zur Vereinfachung die Parameter der Konstruktoren der Klassen `Penguin` und `Ostrich` durch abkürzen.

(3): Die Methode `addAll(Birds)` existiert im Interface `List` und fügt eine gesamte Collection in eine gegebene Collection ein. Die Klasse `ArrayList` implementiert das Interface `List`. Ist die folgende Anweisung – **nach** dem Code der vorherigen Aufgaben – dann zulässig?

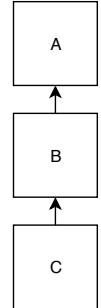
```
new ArrayList<Dodo>().addAll(v);
```

V4 Elemente tauschen

Schreiben Sie eine Methode

```
void switchElements(T[] a, int i, int j) throws IllegalArgumentException
```

Die Methode vertauscht die Elemente im übergebenen Array `a` an den zwei angegebenen Indizes `i` und `j`. Falls für `a` eine `null`-Referenz übergeben wird oder einer der Indizes nicht in dem Array liegt, soll eine `IllegalArgumentException` geworfen werden.

V5 Typhierarchie

Wir betrachten eine Typhierarchie (dargestellt in der Abbildung rechts) mit einer Klasse `A` und einem Erben `B`. Von `B` ist wiederum `C` abgeleitet.

Markieren Sie im folgenden Java-Code jeweils hinter `//`, ob der Compiler die Zeile akzeptiert („Okay“) oder ablehnt („Fehler“). Lösen Sie die Aufgabe zunächst durch eigene Überlegungen und überprüfen Sie erst später mittels Eclipse.

```

1  class A {}
2  class B extends A {}
3  class C extends B {}
4
5  public class G {
6      public void m(List<B> a, List<? extends B> b,
7                  List<? super B> c) {
8
9          a.add(new C()); //
10         b.add(new B()); //
11         c.add(new C()); //
12         a.add(new B()); //
13         b.add(new A()); //
14         c.add(new B()); //
15         a.add(new A()); //
16         b.add(null); //
17         c.add(new A()); //
18         b.add(new C()); //
19
20
21     }
22
23     public static void main(String args[]) {
24         m(new Vector<B>(), new Vector<C>(), new Vector<A>());
25     }
26 }
```

V6 A-well-a bird bird bird, bird is the word Part II ★ ★ ☆

Wir erweitern unsere Typ hierarchie für Vögel aus Aufgabe V3 und betrachten neben den nicht-fliegenden Vögeln nun auch ihre flatternden Artgenossen.

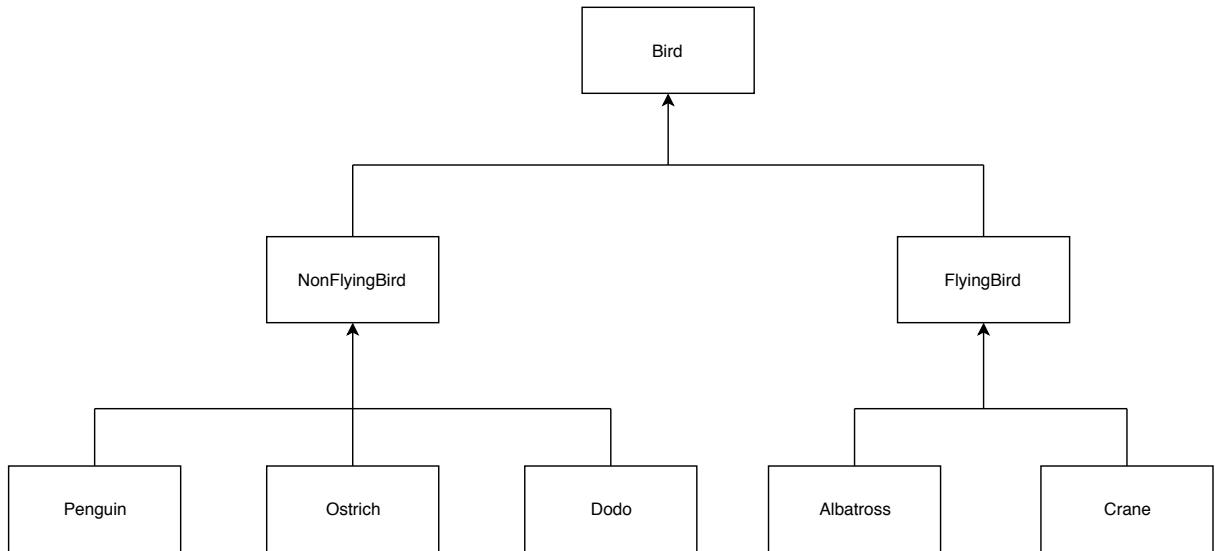


Abbildung 2: Erweiterte Typ hierarchie

Vervollständigen Sie die untenstehenden Deklarationen der Methoden. Dabei sind nur die mit markierten Stellen zu bearbeiten! Geben Sie zu jeder Typangabe eine kurze Erklärung, warum genau diese Typangabe die am besten passende oder korrekte ist. *Ihre Lösungen sollen möglichst weitgehend die Typsicherheit garantieren, aber gleichzeitig flexibel für möglichst viele konkrete Parametertypen sein.*

Es sind immer generische Subtypen zu nutzen.

Hinweis: Zur Vereinfachung wird in den Beispielen nicht auf `null` oder leere Liste getestet.

(1): Die Methode `getFirst (List<.....> aListOfBirds)` liefert den ersten Vogel einer (nicht-leeren) Liste. Das Ergebnis muss kompatibel zum Typ `Bird` sein.

```

Bird getFirst(List<.....> aListOfBirds){
    return aListOfBirds.get(0);
}
  
```

(2): Die Methode `void add(Bird b, List<.....> aListOfBirds)` fügt einen neuen Vogel in die Liste ein. Es sollen dabei Vögel jedes bekannten Typs eingefügt werden können.

```

void add(Bird b, List<.....> aListOfBirds)
    aListOfBirds.add(b);
}
  
```

V7 Array Utility-Klasse



In dieser Aufgabe wollen wir eine bereits vorhandene Utility-Klasse, für Arrays vom Datentyp `int`, so umschreiben, dass diese für jeden beliebigen Datentyp verwendet werden kann. Die Klasse `ArrayUtils` implementiert folgende Methoden:

`void printArray(int[] array)` bekommt ein `int`-Array übergeben und gibt dessen Elemente auf der Konsole aus.

```

1 public static void printArray(int[] array) {
2     for (int i = 0; i < array.length; i++) {
3         System.out.print(array[i]);
4         if (i < array.length - 1) {
5             System.out.print(" ; ");
6         }
7     }
8     System.out.println(); }
```

`int getArrayIndex(int[] array, int value)` bekommt ein `int`-Array und einen Wert übergeben und durchsucht das Array nach dem übergebenen Wert. Wird der Wert gefunden, wird dessen Index im Array zurückgegeben, andernfalls -1.

```

1 public static int getArrayIndex(int[] array, int value) {
2     for (int i = 0; i < array.length; i++) {
3         if (array[i] == value) {
4             return i;
5         }
6     }
7     return -1; }
```

`void simpleSort(int[] array)` sortiert das übergebene `int`-Array in aufsteigender Reihenfolge.

```

1 public static void simpleSort(int[] array) {
2     for (int i = 0; i < array.length; i++) {
3         for (int j = i + 1; j < array.length; j++) {
4             if (array[i] > array[j]) {
5                 int backup = array[i];
6                 array[i] = array[j];
7                 array[j] = backup;
8             }
9     } } }
```

Schreiben Sie nun eine Klasse `GenericArrayUtils`, die alle drei oben genannten Methoden mit einem beliebigen Datentyp `T`, bzw. `T[]` für Arrays, implementiert.

Hinweis: Überlegen Sie sich, wie Sie Elemente vom Typ `T` miteinander vergleichen können, bzw. welche Voraussetzung dieser Typ `T` mit sich bringen muss und wie sich dies im Klassenkopf der zu implementierenden Klasse widerspiegelt.

V8 XYZ

Gegeben seien die folgenden Klassen:

```

1  public class Pair<X, Y> {
2      public X x;
3      public Y y;
4
5      public Pair(X x, Y y) {
6          this.x = x;
7          this.y = y;    }
8      }
9
10 public class Triple<X, Y, Z> {
11     public X x;
12     public Y y;
13     public Z z;
14
15     public Triple(X x, Y y, Z z) {
16         this.x = x;
17         this.y = y;
18         this.z = z;    }
19     }
20
21 public class Utils { }
```

Erweitern Sie nun die Klasse `Utils` um eine `public`-Klassenmethode `intoMap`, ohne dabei den Klassenkopf zu modifizieren. Die Methode bekommt eine `java.util.List` von Tripeln übergeben und gibt eine `java.util.Map` zurück. Jedes Triple in der Liste wird in die Map überführt, indem Sie die `x`-Variable des Triples als Schlüssel verwenden und ein neues Paar aus der `y`- und `z`-Variable des Triples erstellen, um dies als Wert des zugehörigen Schlüssels zu verwenden.

V9 Matrizenmultiplikation Reloaded Reloaded

Ja Sie haben richtig gelesen, die gute, alte Matrizenmultiplikation kommt wieder einmal zurück (diesmal ist auch wirklich das letzte Wiedersehen). Neben Ihrer Implementierung in Racket und Java wollen wir in dieser Aufgabe eine Matrix Klasse implementieren, die es uns erlaubt jeden beliebigen vergleichbaren Datentyp unter Anwendung von Java Generics mit ihr zu verwenden.

Um Ihnen die Aufgabe zu erleichtern, wird ihnen ein Interface bereitgestellt, welches benutzt werden soll um arithmetische Operationen mit den Matrizen durchzuführen. Bevor man einen Datentyp mit unserer Matrix Klasse benutzen kann, muss man zuvor das arithmetische Interface für diesen konkreten Datentyp implementieren. Machen Sie sich mit den Beispielen auf der folgenden Seite vertraut.

Generisches Interface:

```
1 public interface Arithmetic<T> {  
2  
3     /**  
4      * returns the representation of zero  
5      */  
6     T zero();  
7  
8     /**  
9      * Returns the result of the addition of a and b  
10     */  
11    T add(T a, T b);  
12  
13    /**  
14     * Returns the result of the multiplication of a and b  
15     */  
16    T mul(T a, T b);  
17  
18 }
```

Konkrete Implementierung für Gleitkommazahlen mit dem Datentyp Float:

```
1 public class FloatArithmetic implements Arithmetic<Float> {  
2  
3     @Override  
4     public Float zero() {  
5         return 0f;  
6     }  
7  
8     @Override  
9     public Float add(Float a, Float b) {  
10        return a + b;  
11    }  
12  
13     @Override  
14     public Float mul(Float a, Float b) {  
15        return a * b;  
16    }  
17  
18 }
```

Bearbeiten Sie ausgehend davon die Aufgaben auf der nächsten Seite.

V9.1 Erstellen der Matrix-Klasse

Erstellen Sie zunächst die Klasse `public class Matrix<T extends Comparable<T>>`, die die folgenden aufgezählten Variablen besitzt:

- `private Arithmetic<T> arithmetic` ist zuständig für das Durchführen von arithmetischen Operationen.
- `private LinkedList<LinkedList<T>> data` enthält die Daten der Matrix. Hierbei repräsentiert die äußere `LinkedList` die Reihen und die innere `LinkedList` die Spalten der Matrix.
- `private int rows` wird zum speichern der aktuellen Anzahl der Reihen der Matrix verwendet.
- `private int columns` wird zum speichern der aktuellen Anzahl der Spalten der Matrix verwendet.

Implementieren Sie nun folgende Methoden:

- `public Matrix(int rows, int columns, Arithmetic<T> arithmetic)` erhält die gewünschte Größe der Matrix in Form von Reihen und Spalten und ein entsprechendes Objekt dessen Klasse das arithmetische Interface implementiert. Alle übergebenen Variablen dieser Methode sollen in den entsprechenden Klassenvariablen gespeichert werden. Zusätzlich soll jeder Zellenwert mit `arithmetic.zero()` initialisiert werden.
- `public int getRows()` gibt die aktuelle Anzahl der Reihen der Matrix zurück.
- `public int getColumns()` gibt die aktuelle Anzahl der Spalten der Matrix zurück.
- `public T getCell(int row, int column)` erhält den Index einer Reihe sowie den Index einer Spalte und gibt den Wert der Zelle zurück.
- `public void setCell(int row, int column, T value)` erhält den Index einer Reihe sowie den Index einer Spalte und einen gewünschten Wert und setzt diesen an der entsprechenden Stelle in der Matrix ein.

V9.2 Addition und Multiplikation

Implementieren...

... Sie nun die Methode `public Matrix<T> add(Matrix<T> other)`. Diese erhält eine andere Matrix, addiert die übergebene Matrix und die Matrix, auf der die Methode aufgerufen wurde, miteinander und gibt das Ergebnis der Addition zurück. Sollten die Reihen- und/oder Spaltenanzahl der beiden Matrizen nicht übereinstimmen, soll `null` zurückgegeben werden.

... Sie nun die Methode `public Matrix<T> mul(Matrix<T> other)`. Diese erhält eine andere Matrix, multipliziert die Matrix, auf der die Methode aufgerufen wurde, und die übergebene Matrix miteinander und gibt das Ergebnis der Multiplikation zurück. Sollte die Spaltenanzahl der aktuellen Matrix sich von der Reihenanzahl der übergebenen Matrix unterscheiden, soll `null` zurückgegeben werden.

H Achte Hausübung

US-amerikanische Politik

Gesamt 8 Punkte

In dieser achten Hausübung machen wir einen Ausflug in die Politik der Vereinigten Staaten von Amerika. In den beiden Aufgaben werden Sie mittels Generics und Collections zwei völlig unterschiedliche Problemstellungen bearbeiten.

H1 Stringbuilder

0 Punkte

In den Aufgaben H2 und H3 können Sie den sogenannten **StringBuilder** verwenden, dessen Funktionalität schnell erklärt ist. Mithilfe dieser Klasse können Sie schnell und einfach Strings konkatenieren, ohne dabei den **+**-Operator verwenden zu müssen. Dies ist zum einen wesentlich schneller sollten viele Konkatenationen stattfinden und ist zum anderen meist übersichtlicher.

Wir legen zu Beginn immer ein neues Objekt der Klasse **StringBuilder** an. Mittels der **append(String)**-Methode können einfach Strings zum **StringBuilder** hinzugefügt werden. Mit der **toString()**-Methode kann der Builder schließlich in einen String umgewandelt werden.

An einem Beispiel wird dies schnell deutlich:

Variante 1 ohne StringBuilder

```
String s = "";
s = s + "Das " + "ist " + "ein String!";
```

Variante 2 mit StringBuilder

```
StringBuilder sb = new StringBuilder();
sb.append("Das ").append("ist ").append("ein String!");
String s = sb.toString();
```

Für weitere Informationen schauen Sie hier:

<https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html>

Hinweis:

Der String "**\n**" kann für einen Zeilenumbruch verwendet werden!

So wird der Aufruf von

```
System.out.print("Achtung \n Umbruch");
```

auf der Konsole folgendermaßen ausgegeben

```
Achtung
Umbruch
```

H2 Huntington-Hill-Methode**3 Punkte**

Die Vereinigten Staaten von Amerika haben ein Zweikammer-Parlaments-System zu dem neben dem Senat auch das Repräsentantenhaus gehört. Letzteres beinhaltet H Sitze, welche alle 10 Jahre neu unter den Mitgliedsstaaten verteilt werden. Eine der wichtigsten Aufgaben des Repräsentantenhauses ist die maßgebliche Beteiligung an der Gesetzgebung. Es besitzt das alleinige Initiativrecht bei Steuer- und Haushaltsgesetzen und kann außerdem als einzige Institution Amtsenthebungsverfahren gegen Amtsträger einleiten.

Die H Sitze im US-amerikanischen Repräsentantenhaus werden auf die N Staaten mittels der Huntington-Hill-Methode aufgeteilt. Dieses geht wie folgt vor:

1. Zu Beginn erhält jeder Staat initial einen Sitz.
2. Sind noch Sitze übrig? Dann bilden wir für jeden Staat eine Priorität, die sich mittels der Population p und der aktuell zugeteilten Anzahl an Sitzen n des Staates wie folgt berechnet:

$$\frac{p}{\sqrt{n(n+1)}}$$

3. Nun erhält der Staat mit der höchsten Priorität einen Sitz mehr.
4. Solange es noch Sitze zu verteilen gibt, gehe wieder zu 2. zurück. Ansonsten brich ab.

In dieser Aufgabe sollen Sie die Huntington-Hill-Methode umsetzen und dabei Verwendung von Collections machen.

Verbindliche Anforderung: Sie dürfen keine neuen Klassen hinzufügen und dürfen den bereits existierenden Klassen der Vorlage keine Klassenvariablen hinzufügen (dies gilt nur für diese Aufgabe, nicht für die darauffolgende).

Tests:

In dieser Aufgabe sind Ihnen bereits Tests in der Klasse `HuntingtonHillStudentTests` gegeben. Sie können diese beliebig mit Tests Ihrer Wahl ergänzen, sind dazu in dieser Aufgabe allerdings nicht verpflichtet. Zum Testen steht eine Datei `USPopulation.txt` in der Vorlage bereit, welche reale Daten aus dem Jahr 2010 enthält.

H2.1 Staaten- und Fehlerklasse**0 Punkte**

Ergänzen Sie die Klasse `State` um die Methode `public double priority()`, die die Priorität mittels der aktuellen Anzahl an zugewiesenen Sitzen des jeweiligen Staates und der gegebenen Population berechnet.

Ergänzen Sie außerdem die Klasse `MoreStatesThanSeatsException`, welche von `Exception` erbt. Ihr Konstruktor soll den Konstruktor der Oberklasse mit der Nachricht "`There are more States than Seats available!`" aufrufen. Diese Exception wird später geworfen, sollte es mehr Staaten als verteilende Sitze geben. Dies ist nicht möglich, da jeder Staat mindestens einen Sitz erhalten muss.

H2.2 Daten einlesen**1 Punkt**

Ergänzen Sie nun den Konstruktor der Klasse `HuntingtonHill`. Dieser bekommt einen String übergeben, welcher den Dateinamen einer einzulesenden Text-Datei darstellt, sowie die insgesamt zu verteilende Sitzanzahl. In der Text-Datei steht in jeder Zeile der Name eines Staates und seine Population durch ein Semikolon getrennt. Also beispielsweise:

```
Maryland ; 5789929  
Massachusetts ; 6559644  
Michigan ; 9911626  
Minnesota ; 5314879
```

Es wurde bereits ein Attribut `private HashMap<String, State> states` in der Klasse angelegt. Der Konstruktor soll nun jede Zeile der einzulesenden Datei in ein `State`-Objekt und einen String umwandeln. Fügen Sie jedes `State`-Objekt als Wert in die `HashMap` ein, indem Sie den Name des jeweiligen Staates als Schlüssel benutzen.

H2.3 Sitze verteilen**1 Punkt**

Ergänzen Sie nun die Methode `distributeSeats()` der Klasse `HuntingtonHill`, welche eine Exception des Typs `MoreStatesThanSeatsException` werfen kann. Die Methode soll nun die Sitze an die Staaten, welche in der Hashmap gespeichert wurden, nach der Huntington-Hill-Methode verteilen. Sollten mehr Staaten als Sitze vorhanden sein, soll eine entsprechende Exception geworfen werden. Die Anzahl an zu verteilenden Sitzen wurde im Konstruktor übergeben.

H2.4 Ergebnisse printen**1 Punkt**

Ergänzen Sie außerdem die Methode `printDistribution()`, welche einen `String` zurückliefert. Die Verteilung soll immer in folgendem Format zurückgegeben werden, welches anhand des obigen Vier-Staaten Beispiels und 6 zu verteilenden Sitzen verdeutlicht wird:

```
Distributed 6 seats to 4 states.  
  
Massachusetts: 2  
Michigan: 2  
Maryland: 1  
Minnesota: 1
```

Zu Beginn wird also zunächst genannt, wie viele Sitze, an wie viele Staaten verteilt wurden. Nach zwei Absätzen werden dann die Staaten nacheinander aufgelistet mit Ihrer zugewiesenen Sitzanzahl. Die Staaten sind dabei zuerst nach Ihrer Anzahl an Sitzen absteigend und danach lexikographisch aufsteigend geordnet.

H3 Das Sicherheitsprotokoll

5 Punkte

In dieser Aufgabe beschäftigen wir uns mit einer (stark vereinfachten) Modellierung einer Rettungsmission bei akut drohender Gefahr der US-amerikanischen Regierung. Die Grundidee ist dabei, dass die Mitglieder der Regierung gemäß Ihrer Priorität in eine Rangliste geordnet werden und dann in dieser Reihenfolge nacheinander in Sicherheit gebracht werden.

H3.1 Bestandsaufnahme

0 Punkte

Alle Klassen sind bereits vorhanden und müssen nur gegebenenfalls von Ihnen ergänzt werden.

Werfen Sie jedoch einen Blick auf die drei Enum-Typen `SecurityLevel`, `AlertLevel` und `Sex`. In diesen drei Dateien ändern Sie nichts mehr ab!

`Sex` enthält die Geschlechter männlich und weiblich, `SecurityLevel` die Sicherheitsstufen hoch, mittel und niedrig und `AlertLevel` das Bedrohungslevel nach dem Homeland Security Advisory System.

H3.2 Regierungsmitglieder

1 Punkt

Zunächst wollen wir die Regierungsmitglieder modellieren.

Die abstrakte Klasse `GovernmentEmployee` ist bereits angelegt und enthält:

- Die `protected`-Attribute `String name` und `Sex sex`
- Die abstrakten Methoden `SecurityLevel getSecurityLevel()` und `String getTitle()`
- Die bereits implementierte Methode `String toString()`

Ihre Aufgabe ist es nun zunächst die drei Unterklassen `President`, `Secretary` und `Other` zu ergänzen, die alle von `GovernmentEmployee` erben.

Die Konstruktoren der drei Klassen bekommen jeweils einen `String` und ein `Sex` gegeben und setzen diese auf die jeweiligen Attribute der Oberklasse.

Die Methode `getSecurityLevel()` gibt das Sicherheitslevel der Personengruppe zurück, also `SecurityLevel.HIGH` (`President`), `SecurityLevel.MEDIUM` (`Secretary`) und `SecurityLevel.LOW` (`Other`).

Die Methode `compareTo()` erhält jeweils ein `GovernmentEmployee`-Objekt. Hierbei soll gelten: wenn der übergebene Regierungsangestellte ein höheres Sicherheitslevel hat als der aktuelle, wird der Wert -1 zurückgegeben, bei gleicher Priorität 0, ansonsten 1.

Die Methode `getTitle()` liefert die korrekte Anrede zurück. Diese besteht entweder aus `Madame` oder `Mister` zu Beginn, gefolgt von einem eventuellen Titel (bei den Klassen `President` und `Secretary` ist dieser deckungsgleich mit dem Klassennamen, bei der Klasse `Other` entfällt der Titel) und dem Namen des Regierungsmitgliedes.

H3.3 PriorityQueue**1 Punkt**

Eine PriorityQueue ist stets nach den Prioritäten ihrer einzelnen Elemente geordnet. Fügen Sie ein neues Element ein, so rückt es gemäß seiner Priorität an die jeweilige Stelle in der Queue. Die Java Bibliothek bietet bereits eine Klasse PriorityQueue an, wir wollen in dieser Aufgabe jedoch unsere eigene implementieren. Sie dürfen also selbstverständlich andere bereits vorhandenen Implementierungen von Queues (AbstractQueue, ArrayBlockingQueue, ArrayDeque, ConcurrentLinkedDeque, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, SynchronousQueue) **nicht** benutzen. Überlegen Sie sich allerdings, wie Sie die Elemente der Queue speichern wollen. Wir verwenden Generics in dieser Aufgabe, um die Implementierung unabhängig zu gestalten und auch in einem anderen Anwendungsfall als unserem zu verwenden.

Die Klasse soll dabei folgende Methoden bereitstellen:

- Der Konstruktor, welche eine leere Queue erstellt.
- `void enqueue(T e)` soll ein neues Element in die Queue einfügen. Dabei soll die Priorität der Elemente beachtet werden. Dabei steht das Element mit der höchsten Priorität ganz am Anfang der Queue. Bei gleicher Priorität wird das Element als letztes dieser Prioritätsklasse eingefügt, es gibt also kein Vordrängeln. Zum Vergleich dieser Elemente verwenden Sie die Methode `compareTo`.
- `public T dequeue()` entfernt das Element mit der größten Priorität aus der Queue und gibt dieses zurück.
- `int getSize()` gibt die Anzahl der Elemente in der Queue zurück.
- `public String toString()`: liefert einen String, der in absteigend sortierter Reihenfolge alle Elemente der Queue enthält. Dabei soll die Methode `toString()` der Elemente verwendet werden. Jeder Eintrag soll die Form "`#i: xyz\n`" haben, wobei i für die konkrete Position in der Queue (beginnend ab Position 0) und xyz für die Ausgabe der `toString()`-Methode des Elements steht.
- `void addToHeadOfPriorityClass(T e)` schaut ob das übergebene Element bereits in der Queue ist. Wenn ja wird es an der aktuellen Position gelöscht und in seiner jeweiligen Prioritätsklasse an die oberste Stelle gesetzt. Die Queue bleibt also gemäß der Prioritäten geordnet, nur innerhalb einer Prioritätsklasse rückt ein Element nach vorne. Wenn nein, wird das Element neu hinzugefügt, ebenfalls an oberster Stelle der eigenen Prioritätsklasse.

H3.4 EmergencyQueue**1 Punkt**

Ergänzen Sie nun die Klasse `EmergencyQueue<T extends GovernmentEmployee>`, welche bereits ein Attribut `private PriorityQueue<GovernmentEmployee> queue` enthält:

- `public void rescue(int n)`: Die Methode soll die ersten n Mitglieder aus der Queue entfernen und damit in Sicherheit bringen. Geschieht dies soll eine Nachricht auf der Konsole ausgegeben werden, bei dem zuerst der Name über die `getTitle()`-Methode genannt wird und anschließend die Meldung "`was rescued`" folgt. Nach jedem geretteten Mitarbeiter folgt ein Zeilenumbruch.

- Die drei Methoden:

- `public void chooseDesignatedSurvivor(Secretary ds)`: Bestimmt einen Minister als Designated Survivor und gibt ihm damit automatisch die höchste Priorität aller Minister und aktualisiert dementsprechend die Position in der Queue.
- `public void enqueue(GovernmentEmployee newItem)`: Ordnet einen neuen Regierungsvertreter in die Queue ein.
- `public String toString()`: Ruft die `toString()`-Methode des `queue`-Objektes auf.

rufen die jeweiligen Methoden auf dem Attribut `queue` auf.

H3.5 SecurityAgency

1 Punkt

Zum Schluss kümmern wir uns um die Klasse `SecucrityAgency`, welche zuständig für die Kontrolle des Sicherheitszustands der USA ist. Die Klasse besitzt bereits ein `private`-Attribut `al` vom Typ `AlertLevel`, welche die aktuelle Gefährdungslage ausdrückt. Der Konstruktor soll dieses Level bei der Initialisierung zunächst auf `AlertLevel.LOW` setzen. Außerdem ist bereits ein `private`-Attribut `emergency` vom Typ `EmergencyQueue<GovernmentEmployee>` gegeben, sowie ein `private`-Attribut `emps` vom Typ `ArrayList<GovernmentEmployee>`, in dem alle Mitarbeiter gespeichert werden sollen.

Ergänzen Sie die Methode `add(GovernmentEmployee e)`, welche einen neuen Mitarbeiter zu `emps` hinzufügt.

Implementieren Sie die Methode `public void changeAlertLevel(AlertLevel newAL)`, welche das Alarmlevel neu einstuft. Dabei gilt folgendes:

- Beträgt das Level mindestens `AlertLevel.ELEVATED`, so werden alle `Presidents` aus `emps` in die `EmergencyQueue` eingefügt.
- Beträgt das Level mindestens `AlertLevel.HIGH`, so werden alle `Presidents` und `Secretaries` aus `emps` in die `EmergencyQueue` eingefügt.
- Beträgt das Level `AlertLevel.SEVERE`, so werden alle aus `emps` in die `EmergencyQueue` eingefügt.

Sie brauchen hier nur die Hochstufung des Sicherheitslevels zu betrachten, also nur Fälle bei denen von einem niedrigen in ein höheres Sicherheitslevel eingestuft wird.

H3.6 Testen

1 Punkt

Implementieren Sie abschließend eine Testklasse, die das korrekte Verhalten der `EmergencyQueue` und der `PriorityQueue` testet. Dafür wurde bereits eine Klasse `EmergencyTest` angelegt.

Schreiben Sie zu jeder Methode der beiden Klassen mindestens 2 Tests!

Ihnen stehen bereits einige Beispielregierungsmitglieder bereit.

Funktionale und objektorientierte Programmierkonzepte



Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.6

Übungsblatt 9

Themen: Verzeigerte Strukturen

Relevante Folien: Generics und Collections

Abgabe der Hausübung: 18.01.2019 bis 23:55 Uhr

V Vorbereitende Übungen

V1 LinkedList

Für diese Aufgabe betrachten wir folgende Klasse für Listenelemente, die Sie auch schon in der Vorlesung kennengelernt haben.

```
public class ListItem<T>{
    public T key;
    public ListItem<T> next;
}
```

Alle nachfolgenden Aufgaben sollen dabei in folgender Klasse implementiert werden:

```
public class MyLinkedList<T>{

    private ListItem<T> head;

    public MyLinkedList(){
        head = null;
    }

    // Insert your methods here
}
```



Abbildung 1: Eigene LinkedList-Klasse aus der Vorlesung.

V1.1 Neues Element hinzufügen

★ ☆ ☆

Implementieren Sie die Methode `void add(T key)`. Diese bekommt einen neuen Schlüssel übergeben und erstellt ein neues Listenelement mit dem übergebenen Schlüssel, welches ganz am Ende der Liste angehängt wird.

Denken Sie an den Spezialfall, wenn noch kein Element in der Liste vorhanden ist.

V1.2 Element löschen I

★ ☆ ☆

Implementieren Sie die Methode `void delete(int pos)`. Die Methode löscht das Element an der Position `pos` aus der Liste, wobei das erste Element die Position 0 besitzt. Ist `pos` keine gültige Position, so bleibt die Liste unverändert.

V1.3 Element löschen II

★ ☆ ☆

Implementieren Sie die Methode `void delete(T key)`. Die Methode löscht das erste wertgleiche Vorkommen des Elements `key` aus der Liste. Sollte das Element nicht vorkommen, so bleibt die Liste unverändert.

V1.4 Drittletztes Element

★ ★ ☆

Implementieren Sie die Methode `T beforeBeforeLast()`. Der Rückgabewert der Methode ist `null`, falls die Liste nicht mindestens drei Elemente hat. Ansonsten wird der Key vom drittletzten Element der Liste zurückgeliefert.

V1.5 Eins nach links bitte

★ ★ ☆

Implementieren Sie die Methode `void ringShiftLeft()`. Die Methode verschiebt alle Listenelemente um eine Stelle nach links. Das heißt, dass das erste Element das neue letzte Element der Liste wird.

V1.6 Liste in Array

★ ★ ☆

Implementieren Sie (ohne einfach die zugehörige Methode aus der Standardbibliothek aufzurufen) die Methode `T[] listToArray(Class<?> type)`. Die Methode wandelt die Liste in ein Array um, das heißt genau alle Schlüsselwerte der Liste sind in dem Array, welches zurückgegeben wird, in ursprünglicher Reihenfolge enthalten. Sind keine Schlüsselwerte in der Liste enthalten, so soll ein Array der Länge null zurückgegeben werden.

Tipp: Ein Array des Typus T erstellen Sie am besten folgender Maßen:

```
(T[]) Array.newInstance(type, size)
```

V1.7 Liste in Listen

★ ★ ☆

Implementieren Sie die Methode `ListItem<ListItem<T>> listInLists()`. Die Methode teilt die Liste in eine Liste von mehreren einelementigen Listen auf, wobei jeder Schlüsselwert der Eingabeliste, zu genau einem Schlüsselwert einer einelementigen Liste wird.

V1.8 Quadratzahlen aus der Liste entfernen

★ ★ ★

Implementieren Sie die Methode `void deleteSquareNumbers()`. Die Methode entfernt alle Elemente aus der Liste, deren Position in der Liste eine Quadratzahl ist, wobei das erste Listenelement Position 0 hat (was natürlich auch eine Quadratzahl ist).

V1.9 Listen in Liste

★ ★ ★

Implementieren Sie die Methode

```
ListItem<T> listsInList(ListItem<ListItem<T>> lsts).
```

Die Methode erstellt eine zusammenhängende Liste aus dem Parameter `lst`. Dazu sollen alle Listen des Parameters `lst` in der ursprünglichen Reihenfolge hintereinander angefügt werden und der Kopf der resultierenden Liste zurückgegeben werden. Vergleichen Sie dazu auch nochmal Aufgabe V1.7.

V2 Eigene verzeigerte Struktur in Racket

★ ★ ★

In Racket haben Sie Listen schon einige Male gesehen und benutzt. In dieser Aufgabe wollen wir uns nach dem Vorbild von Aufgabe V1 eine eigene verzeigerte Struktur erstellen. Dafür ist bereits folgende Struktur vorgegeben:

```
(define-struct lst-element (value next))
```

V2.1 Sortieren der Liste in aufsteigender Reihenfolge

Definieren Sie eine Funktion `sort-lst`.

Diese bekommt den Kopf einer Liste übergeben, sortiert die `values` der Elemente aufsteigend und liefert den Kopf dieser aufsteigend sortierten Liste zurück. Sie können davon ausgehen, dass die Liste nur Zahlen enthält. Benutzen Sie die Funktion `null?` um zu überprüfen, ob das letzte Element der Liste erreicht wurde. In diesem Fall gibt die Funktion `null?` dann `true` zurück, wenn damit das `next`-Feld, der `lst-element`-Struktur überprüft wird.

V3 Alternative LinkedList

In der Vorlesung haben Sie außerdem eine alternative `LinkedList` Implementierung kennengelernt. Diese zeichnet sich dadurch aus, dass anstelle eines `Keys` pro Item der Liste, ein Array von `Keys` mit einer vorher festgelegten Größe verwendet wird. Wir nennen diese Art von Listen in dieser Aufgabe `ArrayList`.

Gegeben sei dafür folgende Klasse für Listenelemente:

```
public class ArrayListItem<T>{
    public T[] a;
    public int n;
    public ArrayListItem<T> next;
}
```

Alle nachfolgenden Aufgaben sollen dabei in folgender Klasse implementiert werden:

```
class ArrayList<T>{

    private ArrayListItem<T> head;
    private int N; // size of each array stored in the items

    public ArrayList(int arraySize){
        head = null;
        this.N = arraySize;
    }

    // Insert your methods here

}
```

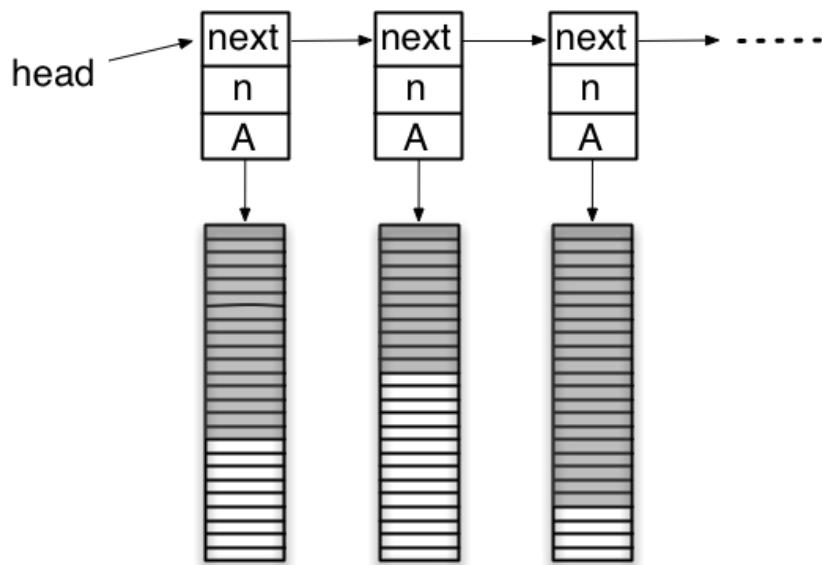


Abbildung 2: Implementierung einer eigenen ArrayList-Klasse aus der Vorlesung.

Hinweis: Wenn von Indizes die Rede ist, dann sind diese durchlaufend. Bedeutet bei Arrays der Länge n liegt der Index $n + m$ im zweiten Array, solange $m < n$ gilt.

V3.1 contains-Methode



Implementieren Sie die Methode `int contains(T e)`. Diese durchsucht die Liste nach dem übergebenen Element `e` und gibt den ersten gefundenen Index in der Liste zurück, sofern es dort enthalten ist. Ist das übergebene Element nicht enthalten, soll stattdessen `-1` zurückgegeben werden.

V3.2 get-Methode

★ ☆ ☆

Implementieren Sie die Methode `T get(int index)`. Diese gibt das Element an dem übergebenen Index in der Liste zurück. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner null ist oder der Index die Größe der Liste überschreitet.

V3.3 set-Methode

★ ★ ☆

Implementieren Sie die Methode `void set(T e, int i)`. Diese bekommt ein Element `e` vom Typ `T` und einen Index `i` übergeben und ersetzt das aktuelle Element am Index der Liste mit dem übergebenen. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner 0 ist oder der Index die Größe der Liste überschreitet.

V3.4 remove-Methode

★ ★ ☆

Implementieren Sie die Methode `void remove(int i)`. Diese bekommt einen Index `i` übergeben und löscht das Element am übergebenen Index in der Liste. Alle nachfolgenden Elemente der Liste müssen daher um einen Index nach vorne verschoben werden. War das zu löschen Element das letzte Element in dem Array seines `ArrayListItems`, so muss der Verweis auf dieses `ArrayListItem` auf `null` gesetzt werden, da es nicht mehr verwendet wird. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner 0 ist oder der Index die Größe der Liste überschreitet.

V3.5 Komprimierung

★ ★ ★

Implementieren sie die Methode `void compress()`, welche die Liste komprimiert. Das heißt nach Aufruf der Methode müssen alle internen Arrays bis auf das Letzte komplett gefüllt sein. Das letzte Array muss hierbei mindestens ein Element ungleich `void` enthalten, das heißt es muss $n > 0$ gelten.

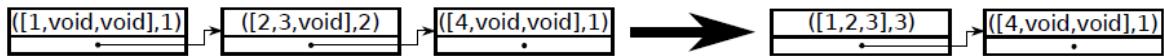


Abbildung 3: Beispiel für `compress` mit $N = 3$

H Neunte Hausübung

Selbstorganisierende verzeigerte Strukturen

Gesamt 10 Punkte

In dieser neunten Hausübung beschäftigen Sie sich wieder mit verzeigerten Strukturen. Die Besonderheit diesmal: Es handelt sich um sogenannte selbstorganisierende, verzeigte Strukturen. Diese Strukturen ordnen ihre Elemente mittels einer Organisationsstrategie selbst um, um die Suche nach einzelnen Elementen effizienter zu gestalten.

H1 Selbstorganisierende Liste

4 Punkte

Aus der Vorlesung kennen Sie bereits die Datenstruktur `LinkedList`, in der die Suche nach einem Element in einer n -elementigen `LinkedList` im schlimmsten Fall n Vergleiche benötigt. Das Ziel einer selbstorganisierenden Liste ist es die Elemente, die häufiger in der Liste gesucht werden, weiter an den Anfang der Liste zu verschieben, um somit die Suche nach eben diesen Elementen effizienter zu gestalten. In dieser Aufgabe werden Sie eine solche Listen-Klasse mit drei verschiedenen Organisationsstrategien mittels der Ihnen bekannten Klasse `ListItem<T>` aus den vorherigen vorbereitenden Übungen sowie der Vorlesung, implementieren.

Machen Sie sich zunächst mit den gegebenen Klassen aus der Vorlage vertraut. Die Klasse `ListItem<T>` sollte Ihnen bekannt sein. Im `Enum`-Typ `ReorganizingAlgorithm` sind unsere drei verschiedenen Organisationsstrategien definiert, später dazu mehr. Die Klasse `SimpleLinkedList<T>` stellt eine sehr vereinfachte `LinkedList` mit nur wenigen Methoden dar, weitere Methoden werden wir für diese Aufgabe auch nicht benötigen. Machen Sie sich insbesondere mit der Methode `public T search(Predicate<T> predicate)` vertraut. Diese bekommt ein Prädikat übergeben und gibt den Key des ersten Listenelements in der Liste zurück, bei der die Methode `test` des Prädikats `true` zurück liefert wenn der Key dieses Listenelements übergeben wird. Existiert kein solcher Key so gibt die Methode `null` zurück.

Nun wollen wir damit anfangen unsere selbstorganisierende Liste zu implementieren. Es ist bereits eine Klasse `SelfOrganizingLinkedList` angelegt, welche von der Klasse `SimpleLinkedList` erbt. Die Klasse ist, wie auch ihre Oberklasse, generisch mit `T` parametrisiert. Der Konstruktor der Klasse hat einen Parameter vom `Enum`-Typ `ReorganizingAlgorithm`, dieser bestimmt die verwendete Organisationsstrategie der Liste, welche Sie in den nachfolgenden Aufgaben implementieren.

Verbindliche Anforderungen für die Aufgaben H1.1, H1.2 und H1.3:

- Bei jedem Aufruf der `search`-Methode darf die Liste nur einmal durchlaufen werden.
- Es dürfen keine Objekte mittels `new` eingerichtet werden.
- Für das Umordnen der Liste dürfen Sie nur die Zeiger, also das Attribut `next`, der Listenelemente sowie den Kopf der Liste verwenden.
- Das Attribut `key` der Listenelemente darf nicht überschrieben werden.
- Die Klassen `SimpleLinkedList` und `ListItem` der Vorlage dürfen in keiner Weise modifiziert werden.

Hinweis: head ist ein ganz normales Listenelement! Es besitzt also genauso key und next-Attribut, wie jedes andere Element auch.

H1.1 Strategie *Move to Front*

1 Punkt

Überschreiben Sie die Methode `public T search(Predicate<T> predicate)` der Klasse `SelfOrganizingLinkedList`, sodass die Strategie *Move to Front* umgesetzt wird, sofern der im Konstruktor übergebene Parameter `ReorganizingAlgorithm.MOVETOFRONT` ist.

Diese läuft folgendermaßen ab: Befindet sich der gesuchte Key in der Liste, so wird dieses Element der neue Kopf der Liste (siehe Beispiel in Abbildung 4).

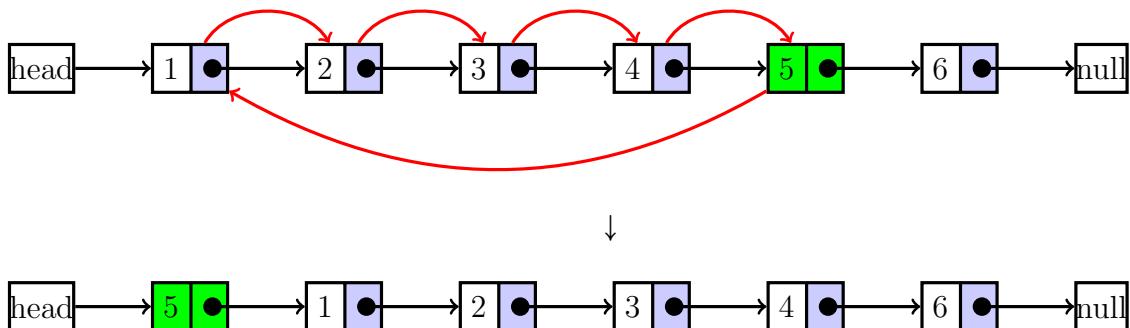


Abbildung 4: Beispiel der *Move to Front* Strategie für die Suche nach dem Key 5

H1.2 Strategie *Transpose*

1 Punkt

Überschreiben Sie die Methode `public T search(Predicate<T> predicate)` der Klasse `SelfOrganizingLinkedList`, sodass die Strategie *Transpose* umgesetzt wird, sofern der im Konstruktor übergebene Parameter `ReorganizingAlgorithm.TRANSPOSE` ist.

Diese läuft folgendermaßen ab: Befindet sich der gesuchte Key in der Liste, so wird das gefundene Element mit dessen Vorgänger in der Liste vertauscht (siehe Beispiel in Abbildung 5).

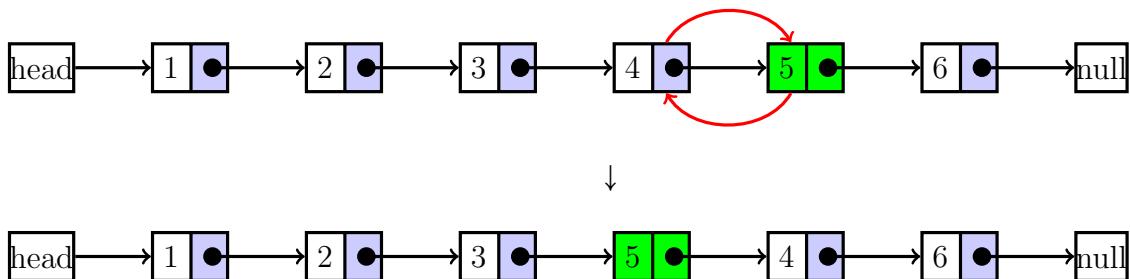


Abbildung 5: Beispiel der *Transpose* Strategie für die Suche nach dem Key 5

H1.3 Strategie *Count***2 Punkte**

Modifizieren Sie nun die Methode `public T search(Predicate<T> predicate)` der Klasse `SelfOrganizingLinkedList`, sodass die Strategie *Count* umgesetzt wird, sofern der im Konstruktor übergebene Parameter `ReorganizingAlgorithm.COUNT` ist.

Diese läuft folgendermaßen ab: Bei dieser Strategie wird für jedes Element der Liste gezählt, wie oft es gesucht wurde. Die Liste ist nach jeder Suche absteigend nach der Anzahl der Suchanfragen der Elemente sortiert (siehe Beispiel in Abbildung 6). Speichern Sie die Anzahl der Suchanfragen eines Listenelements in dem `public`-Attribut `counter` des jeweiligen Listenelements ab.

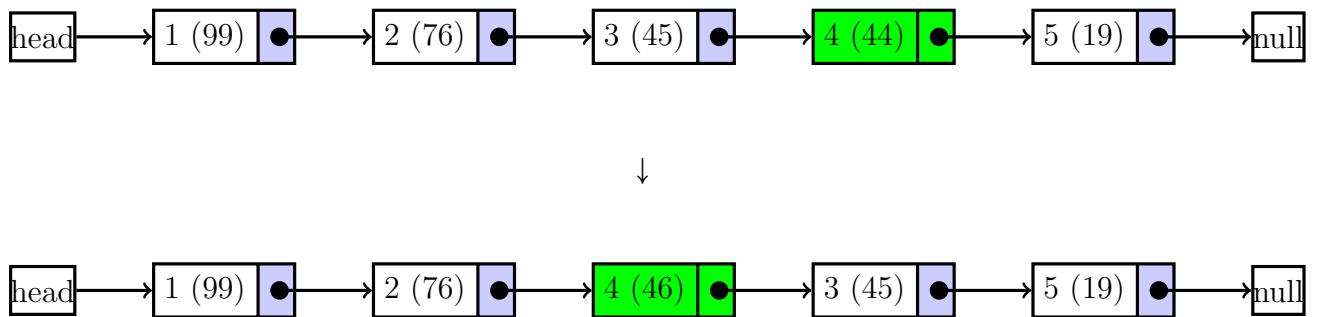


Abbildung 6: Beispiel der *Count* Strategie für die zweifache Suche nach dem Key 4

H2 Selbstorganisierender Baum**6 Punkte**

In der ersten Hausübung haben Sie Binäräbäume kennengelernt. Ähnlich wie in Aufgabe H1 erweitern wir diese Binäräbäume so, dass Sie sich selbst organisieren. Die Besonderheit dabei ist, dass häufig angefragte Elemente in die Nähe der Wurzel gebracht werden.

Erinnern Sie sich (oder schauen Sie erneut auf Übung 1 nach), wie die Eigenschaft eines Binärbaums definiert war: Bei jedem Knoten sind im linken Teilbaum kleinere und im rechten Teilbaum größere Werte zu finden. In Abbildung 7 sehen Sie nochmal ein Beispiel für diese Datenstruktur.

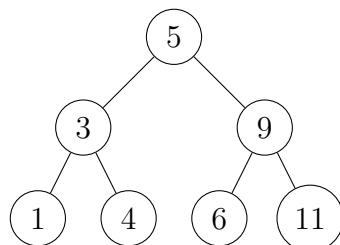


Abbildung 7: Ein Beispielbinärbaum mit Suchbaumeigenschaft

Es sind bereits die Klassen `BinaryTreeNode` und `BinaryTree` angelegt. Ein Knoten besteht (wie Sie es auch schon kennen) wieder aus einem Wert und dem linken, sowie rechten Nachfolger. Die Methoden zum Einfügen und Suchen von Werten im Binärbaum sind bereits implementiert worden. Außerdem ist bereits eine Klasse

```
SelfOrganizedTree<T extends Comparable<T>> extends BinaryTree<T>
```

erstellt. Ihre Aufgabe ist es nun mithilfe der folgenden Teilaufgaben, Schritt für Schritt die dortige Methode `boolean search(T value)` so zu erweitern, dass die beschriebenen Umordnungsstrategien umgesetzt werden.

Zur Wiederholung und Übersicht nochmals die wichtigsten Begriffe für diese Übung:

- **Wurzel:** Der Knoten ohne Vorgänger und damit zentrales Element des Baumes.
- **Kindknoten von X:** Der direkte, entweder linke oder rechte Nachfolger von X. Äquivalent sind Enkelknoten die Nachfolger der Nachfolger von X.
- **Elterknoten von X:** Der direkte Vorgänger von X. Äquivalent sind Großelterknoten die direkten Eltern der Eltern und Urgroßelterknoten die direkten Eltern der Eltern der Eltern.

Verbindliche Anforderungen für die Aufgaben H2.1, H2.2 und H2.3:

- Bei jedem Aufruf der `search`-Methode darf der Baum nur einmal durchlaufen werden.
- Es dürfen keine Objekte mittels `new` eingerichtet werden.
- Für das Umordnen des Baums dürfen Sie nur die Zeiger, also die Attribut `left` und `right`, der Knoten sowie die Wurzel des Baums verwenden.
- Das Attribut `value` der Knoten darf nicht überschrieben werden.
- Die Klassen `BinaryTree` und `BinaryTreeNode` der Vorlage dürfen in keiner Weise modifiziert werden.

H2.1 Zick-Rotation und Zack-Rotation

2 Punkte

Sollte `value` ein Kindknoten der Wurzel sein, so wird entweder eine sogenannte Zick- oder Zack-Rotation durchgeführt. Dabei rutscht `value` in die Wurzel und die entsprechenden Teilbäume werden angepasst. Dabei unterscheiden wir zwei Fälle:

1. Der Knoten mit `value` ist der linke Kindknoten der Wurzel. Dann wird eine Zick-Rotation (oder Rechtsrotation) durchgeführt. Ein Beispiel hierfür finden Sie in Abbildung 8.
2. Der Knoten mit `value` ist der rechte Kindknoten der Wurzel. Dann wird eine Zack-Rotation (oder Linksrotation), analog zur Zick-Rotation, durchgeführt.

Beachten Sie: Der Baum wird auf der Kante zwischen X und P rotiert.

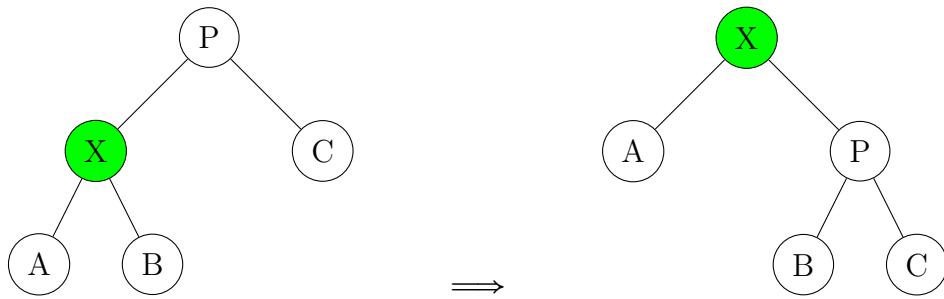


Abbildung 8: Beispiel für eine Zick-Rotation, wenn X der gesuchte Wert ist.

H2.2 Zick-Zick-Rotation und Zack-Zack-Rotation**2 Punkte**

Im Gegensatz zur Zick- oder Zack-Rotation ist der gesuchte Knoten nicht das direkte Kind der Wurzel. Hierbei tauscht der gesuchte Knoten die Position mit seinem Großeltern und alle weiteren Teilbäume werden entsprechend gesetzt. Auch hier werden wieder zwei Fälle unterschieden:

1. Der Knoten mit `value` ist der linke Kindknoten seines Elterknoten, welcher das linke Kind des Großelters ist. Dann wird eine Zick-Zick-Rotation (oder zweifache Rechtsrotation) durchgeführt. Ein Beispiel hierfür finden Sie in Abbildung 9.
2. Der Knoten mit `value` ist der rechte Kindknoten seines Elterknoten, welcher das rechte Kind des Großelters ist. Dann wird eine Zack-Zack-Rotation (oder zweifache Linksrotation), analog zur Zick-Zick-Rotation, durchgeführt.

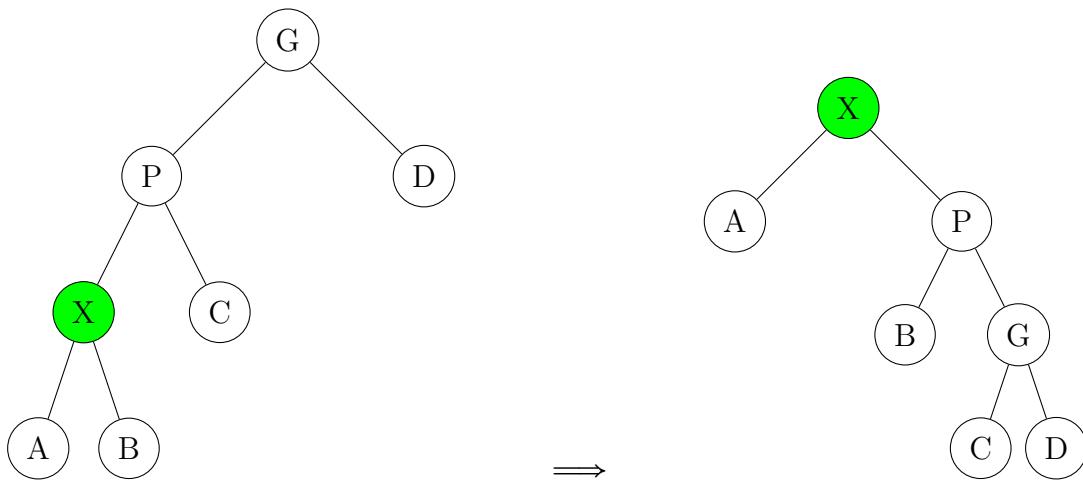


Abbildung 9: Beispiel für eine Zick-Zick-Rotation, wenn X der gesuchte Wert ist.

Beachten Sie: Der Baum wird zuerst auf der Kante, die P mit seinem Elterknoten G verbindet, und danach auf der Kante, die X mit dem Elterknoten P verbindet, rotiert.

H2.3 Zack-Zick-Rotation und Zick-Zack-Rotation**2 Punkte**

Die letzten Rotationen laufen ähnlich zu denen aus Aufgabe H2.2. Der gesuchte Knoten ist wieder nicht das direkte Kind der Wurzel. Hierbei tauscht der gesuchte Knoten die Position mit seinem Großeltern und alle weiteren Teilbäume werden entsprechend gesetzt. Auch hier werden wieder zwei Fälle unterschieden:

1. Der Knoten mit `value` ist der rechte Kindknoten seines Elterknoten, welcher das linke Kind des Großelters ist. Dann wird eine Zack-Zick-Rotation (oder Links-Rotation gefolgt von einer Rechts-Rotation) durchgeführt. Ein Beispiel hierfür finden Sie in Abbildung 10.
2. Der Knoten mit `value` ist der linke Kindknoten seines Elterknoten, welcher das rechte Kind des Großelters ist. Dann wird eine Zick-Zack-Rotation (oder Rechts-Rotation gefolgt von einer Links-Rotation), analog zur Zack-Zick-Rotation, durchgeführt.

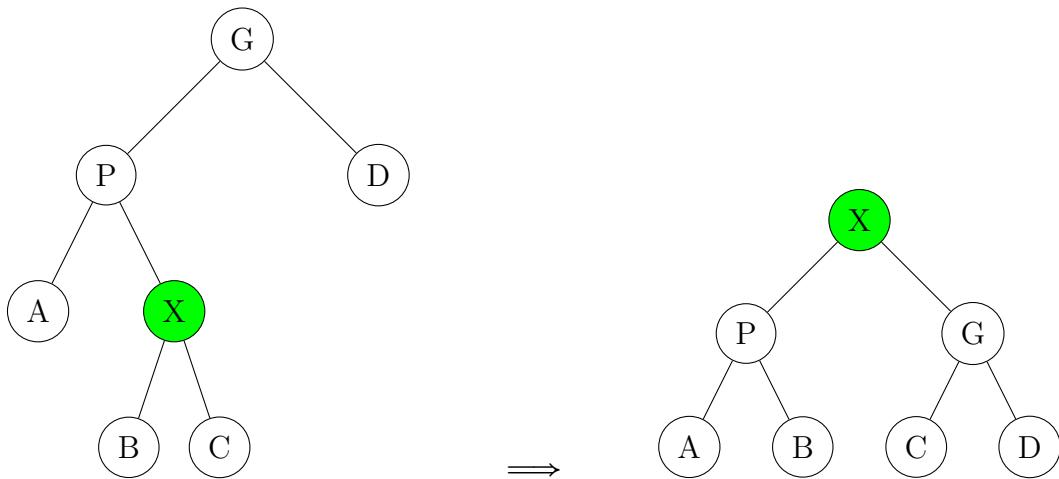


Abbildung 10: Beispiel für eine Zack-Zick-Rotation, wenn X der gesuchte Wert ist.

Beachten Sie: Der Baum wird zuerst auf der Kante, die X mit seinem Elterknoten P verbindet, und danach auf der Kante, die X mit dem Elterknoten G verbindet, rotiert.