SE1, Aufgabenblatt 7

Softwareentwicklung I – Wintersemester 2011/12

Rekursion, Reguläre Ausdrücke

MIN-CommSy-URL: ..... https://www.mincommsy.uni-hamburg.de/
CommSy-Projektraum ..... SE1 CommSy WiSe 11/12
Ausgabewoche ..... 01. Dezember 2011

Kernbegriffe

Zeichenketten bzw. kurz (aus dem Englischen) *Strings* werden in Java als Exemplare der Klasse `java.lang.String` realisiert. Nach der Erzeugung ist ein `String`-Objekt nicht mehr veränderbar. Alle Methoden, die den String scheinbar verändern, liefern in Wirklichkeit ein neues `String`-Objekt zurück und lassen den Original-String unberührt. Daher können Referenzen auf `String`-Objekte bedenkenlos weitergegeben werden. `String`-Objekte können auch durch *String-Literale* (Zeichenfolgen zwischen doppelten Anführungsstrichen) erzeugt werden. Da String-Variablen Referenzen auf `String`-Objekte sind, muss der Vergleich zweier `String`-Referenzen vom Vergleich zweier `String`-Objekte über die Methode `equals` unterschieden werden.

Die Klasse `String` bietet mit der Methode `boolean matches(String regex)` seit Java 1.4 die Möglichkeit des *Pattern Matching* über *reguläre Ausdrücke*.

Neben den formalen Voraussetzungen für die Übersetzbarkeit von Quelltexten (siehe Aufgabenblatt 3) gibt es Richtlinien, die nicht durch die Syntax einer Sprache vorgeschrieben werden. Einige sind grundsätzlich geltende *Konventionen*, wie z.B., dass in Java Klassennamen groß und Variablennamen sowie Methodennamen klein geschrieben werden. Diese Konventionen wurden durch die Firma Sun Microsystems vorgegeben und sollten bei der Softwareentwicklung mit Java eingehalten werden. Andere Konventionen sind projektspezifisch und legen z.B. die Formatierung (Einrückungen etc.) von Java-Quelltext fest. Auch für SE1 gelten solche Quelltext-Konventionen, die eingehalten werden sollten.

Wiederholungen können alternativ auch mit Hilfe von *Rekursion* (engl.: recursion) definiert werden. Rekursion bedeutet Selbstbezüglichkeit (von lateinisch recurrere = zurücklaufen). Sie tritt immer dann auf, wenn etwas auf sich selbst verweist. In Java kann eine Methode sich selbst aufrufen. Ein rekursiver Aufruf kann direkt erfolgen (direkte Rekursion), oder auch über mehrere Zwischenschritte entstehen (indirekte Rekursion).

Die Entscheidung, ob bei einer bestimmten Problemstellung Rekursion oder Iteration einzusetzen ist, hängt im Wesentlichen davon ab, welche der alternativen Varianten lesbarer und verständlicher ist und welcher Rechen- und Speicheraufwand jeweils mit ihnen verbunden ist.

Der Speicher für die Variablen und Daten eines Java-Programms teilt sich in zwei Bereiche: den *Aufruf-Stack* (engl.: call stack) und den *Heap*. Auf einem allgemeinen Stack werden Elemente nach dem LIFO (Last In First Out) Prinzip verwaltet, d.h. zuletzt abgelegte Elemente werden zuerst wieder entnommen. Der Aufruf-Stack dient zum Ablegen von lokalen Variablen. Bei jedem Methodenaufruf werden die Parameter und alle lokalen Variablen auf den Aufruf-Stack geschrieben und erst beim Zurückkehren, z.B. mittels `return`, wieder entfernt. Die maximale Größe des Aufruf-Stacks (*Stack-Limit*) legt fest, wie tief geschachtelt Methodenaufrufe seien dürfen; dies kann bei aufwändigen rekursiven Berechnungen eine Rolle spielen. Wird das Stack-Limit erreicht, bricht die Ausführung mit einer `StackOverflowException` ab. Der Heap eines Java-Programms hingegen dient zum Verwalten der während der Programmausführung erzeugten Objekte. Jedes mittels `new` erzeugte Objekt wird im Heap gespeichert und verbleibt dort mindestens so lange, bis es im Programm keine Referenzen mehr auf das Objekt gibt. Die maximale Heap-Größe begrenzt die Anzahl der Objekte, die in einer Anwendung gleichzeitig existieren können. Wird bei einem bereits vollen Heap ein weiteres Objekt erzeugt, bricht die Ausführung mit einer `OutOfMemoryException` ab.

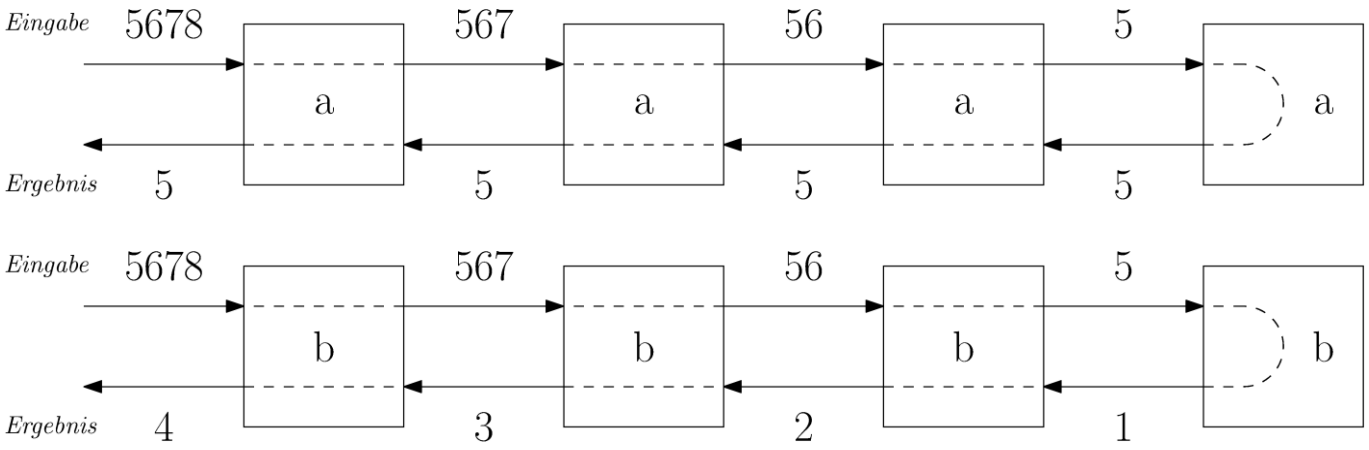
Lernziele

Rekursion verstehen, Zeichenketten und einfache reguläre Ausdrücken benutzen können.

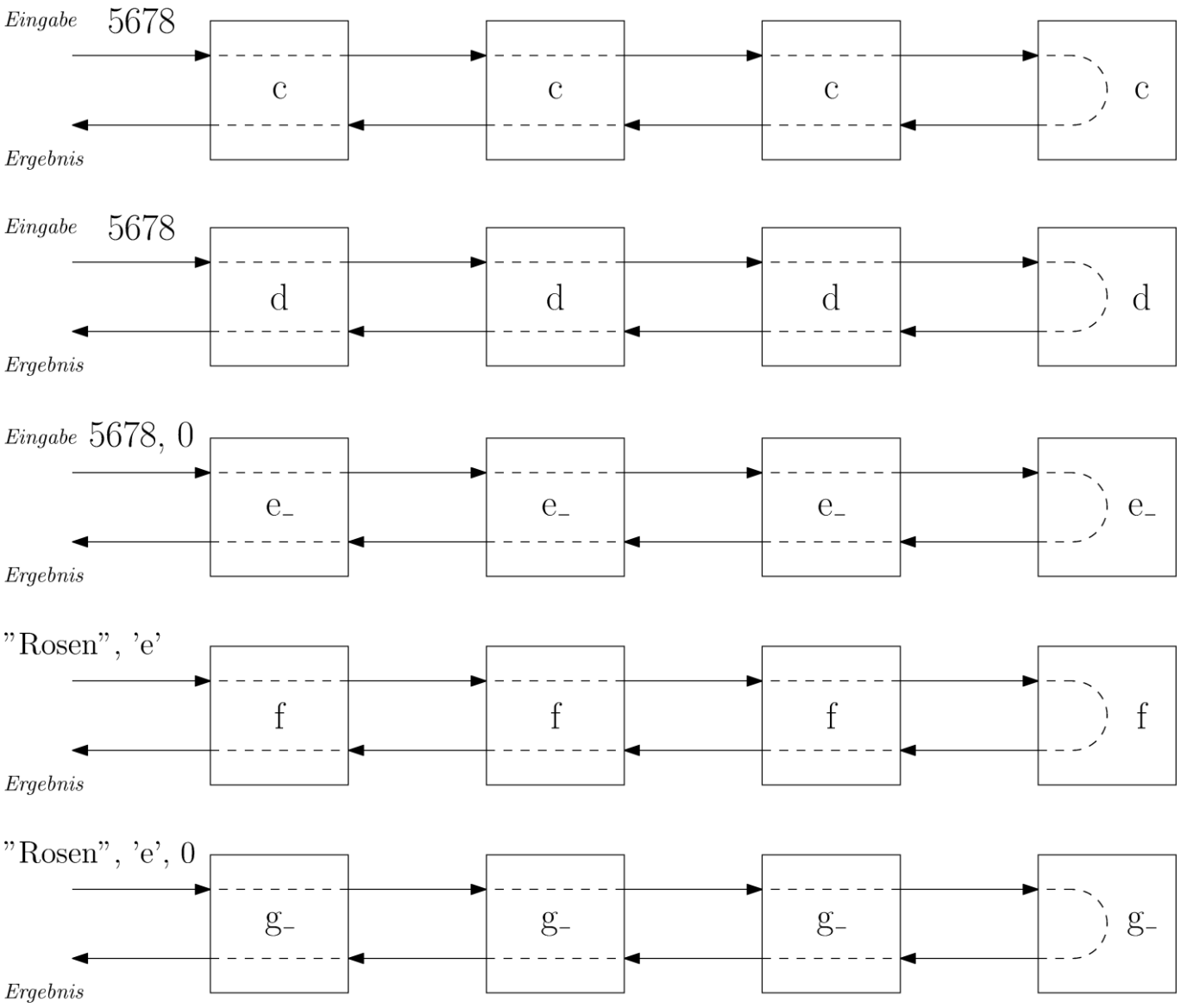
Aufgabe 7.1 Rekursion konsumieren

Die Klasse *Konsumieren* beinhaltet zahlreiche Methoden, in denen rekursive Algorithmen umgesetzt werden. Die Funktionen `a`, `b`, `c`, `d` und `f` sind rekursive Methoden, da sie sich selbst aufrufen. Die zwei Methoden `e` und `g` sind dagegen nicht rekursiv. Sie starten jedoch einen rekursiven Prozess, indem sie die rekursiven Hilfsmethoden `e_` bzw. `g_` aufrufen. (Der Unterstrich ist lediglich eine Konvention und hat keine tiefere Bedeutung. Die Methoden könnten auch ganz anders heißen.)

Der Datenfluss durch die Aktivierungen der Methoden kann für konkrete Beispieleingaben anhand von Diagrammen veranschaulich werden. Für die Aufrufe von `a` und `b` in `testeAlleMethoden` sehen diese wie folgt aus:



7.1.1 Vervollständigt **mindestens vier** der folgenden Diagramme für die verbleibenden fünf Methoden:



7.1.2 Kommentiert die Methoden im Quelltext, damit ein Klient weiß, was diese Methoden leisten.

Aufgabe 7.2 Rekursion produzieren

Füllt die Rümpfe der Methoden der Klasse *Produzieren* mit rekursiven Algorithmen. Hier sind also keine Schleifen (for, while, do-while) erlaubt! Die kommentierten Beispielauswertungen veranschaulichen mögliche Lösungen.

Für eine erfolgreiche Abnahme reichen **vier Methoden** aus, ihr könnt also eine schwierige Methode auslassen.

Aufgabe 7.3 Reguläre Ausdrücke

7.3.1 Öffnet das Projekt *Regex* und erzeugt ein neues Exemplar der Klasse `RegexGUI`. Macht euch mit den Elementen der Oberfläche vertraut. Ändert den Text in den beiden Eingabefeldern, die mit *input* und *regex* bezeichnet sind, und beobachtet, wie sich die GUI daraufhin ändert.

7.3.2 Was bedeuten die folgenden regulären Ausdrücke?

[a-z]
[a-z]+
.
\.
[a\-z]

Probiert es aus oder schaut euch die Dokumentation im Web an, unter *Summary of regular-expression constructs* auf der Seite http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

7.3.3 In SE1 gibt es Quelltextkonventionen für die Benennung von Klassen, Methoden und Feldern. Entwerft je einen regulären Ausdruck für gültige Klassennamen, Methodennamen und Feldnamen. Macht euch dazu klar, wie diese Namen aufgebaut sind. Wo dürfen innerhalb der Namen Kleinbuchstaben, Großbuchstaben, Ziffern und Unterstriche verwendet werden?

7.3.4 Der gegebene reguläre Ausdruck für eMail-Adressen ist bisher sehr restriktiv. Erweitert ihn, so dass auch die folgenden eMail-Adressen erkannt werden:

max.mustermann@gmx.de
chunkylover53@aol.com
musterma@informatik.uni-hamburg.de
Veni_Vidi_Vici@2isnot3.eu

Hinweise: Unter *Predefined character classes* werden einige nützliche Abkürzungen aufgelistet.

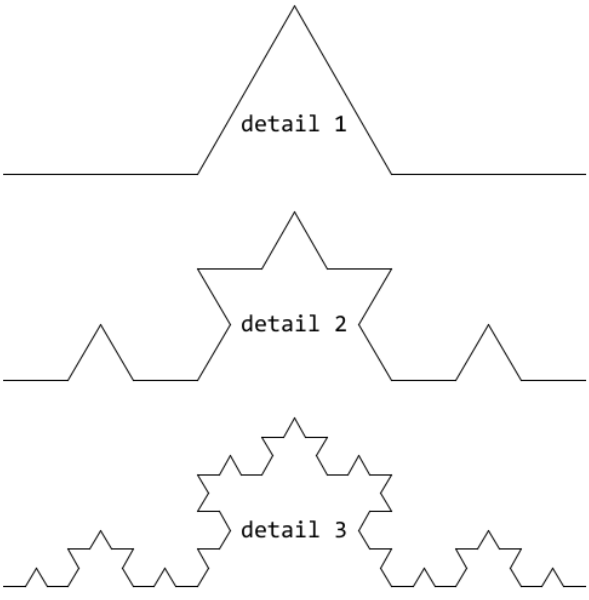
7.3.5 *Zusatzaufgabe*: Entwerft einen regulären Ausdruck für gültige IP-Adressen, also vier Zahlen von 0 – 255, die durch drei Punkte voneinander getrennt sind (zum Beispiel `192.168.0.1`). Die eigentliche Herausforderung dabei ist es, einen regulären Ausdruck für Zahlen von 0 – 255 zu finden.

Testet auf jeden Fall die Randfälle 0, 9, 10, 99, 100, 199, 200, 249, 250, 255.

Hinweise: Die Disjunktion zweier Ausdrücke geschieht mittels |, wobei der linke Teilausdruck bevorzugt wird. Der reguläre Ausdruck `[0-255]` wird hier übrigens nicht funktionieren. (Er bedeutet 0, 1, 2 oder 5.)

Aufgabe 7.4 Turtle Graphics

7.4.1 Implementiere in `Dompteur` die Methode `zeichneKochKurve`, die eine Koch-Kurve mit einem gewissen Detailgrad zeichnet. Die ersten drei Detailgrade sind in der folgenden Grafik veranschaulicht:



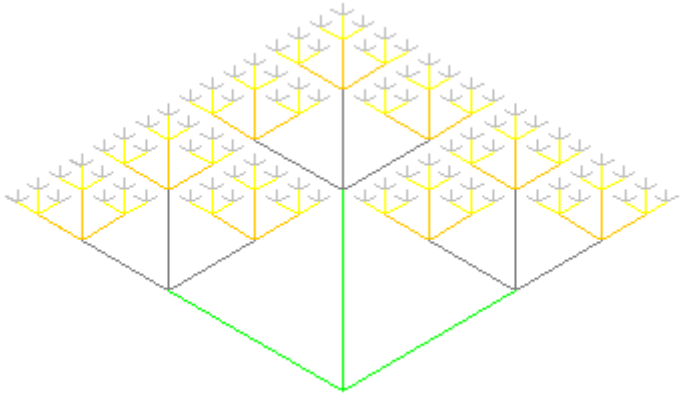
Wie man sehen kann, taucht die Detail-1-Kurve in verkleinerter Version 4x in der Detail-2-Kurve auf, und zwar auf 1/3 verkleinert. Die Detail-2-Kurve taucht wiederum in 1/3-Größe 4x in der unteren auf usw.

Allgemein formuliert: um eine Detail-n-Kurve der Länge L zu zeichnen, muss man 4 Detail-(n-1)-Kurven der Länge L / 3 zeichnen. Es bietet sich also an, `zeichneKochKurve` rekursiv zu implementieren:

Wenn der Detailgrad D bei 0 angekommen ist, zeichnet man einfach eine Linie der Länge L. Ansonsten:

- Zeichne Koch-Kurve der Länge L / 3 mit Detailgrad D - 1
- Drehe um 60 Grad nach links (negativ)
- Zeichne Koch-Kurve der Länge L / 3 mit Detailgrad D - 1
- Drehe um 120 Grad nach rechts (positiv)
- Zeichne Koch-Kurve der Länge L / 3 mit Detailgrad D - 1
- Drehe um 60 Grad nach links (negativ)
- Zeichne Koch-Kurve der Länge L / 3 mit Detailgrad D - 1

7.4.2 Schreibt eine Methode in `Dompteur`, die einen Baum zeichnet. Per Parameter soll die Anzahl der Äste, die aus jeder Gabelung entstehen, festgelegt werden. Auch soll die Position, an der der Baum gezeichnet wird, und die Länge der ersten Äste angegeben werden können. Die Äste sollen pro Gabelung kürzer werden, bis ein Minimalwert erreicht ist. Zusätzlich soll die Farbe der Äste variieren.



Hinweis: Implementiert die Methode rekursiv, so dass für jede Gabelung ein neuer Methodenaufruf erfolgt. Als Abbruchbedingung der Rekursion kann das Unterschreiten einer bestimmten Astlänge dienen.