SE1, Aufgabenblatt 8

Softwareentwicklung I – Wintersemester 2016/17

Rekursion

Moodle-URL: uhh.de/se1

Feedback zum Übungsblatt: uhh.de/se1-feedback

Projektraum ..... Softwareentwicklung 1- WiSe 2016/17

Ausgabewoche ..... 8. Dezember 2016

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.

Wiederholungen können alternativ zur Iteration 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.

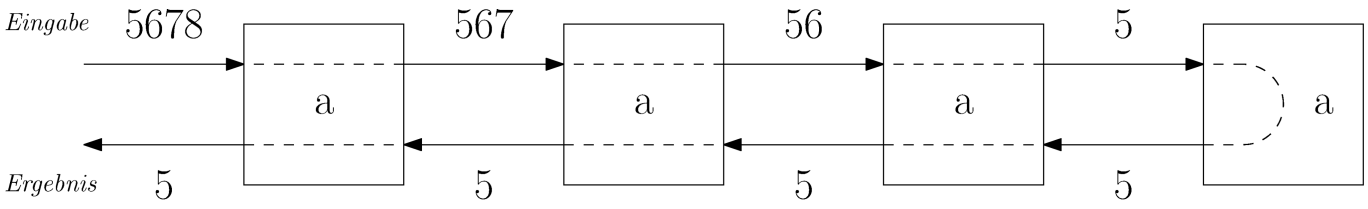
Lernziele

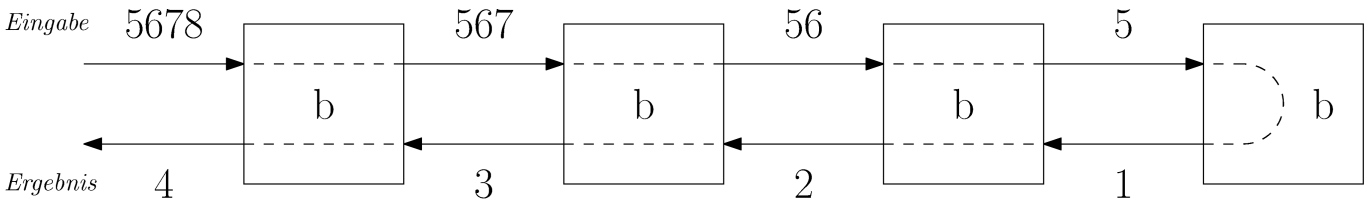
Rekursion verstehen.

Aufgabe 8.1 Vorgegebene Rekursionen analysieren

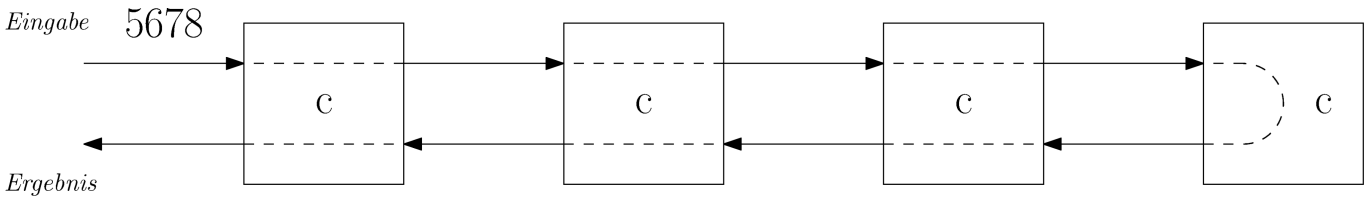
Die Klasse *Analysieren* des Projekts *Rekursion* beinhaltet zahlreiche Methoden, in denen rekursive Algorithmen umgesetzt werden. Die Methoden `a`, `b`, `c`, `d` und `f` sind rekursive Methoden, da sie sich selbst aufrufen. Die zwei Methoden `e` und `g` sind dagegen nicht rekursiv. Sie starten jedoch einen rekursiven Prozess, indem sie rekursive Hilfsmethoden `e_` bzw. `g_` aufrufen. (Der Unterstrich hat übrigens keine tiefere Bedeutung.)

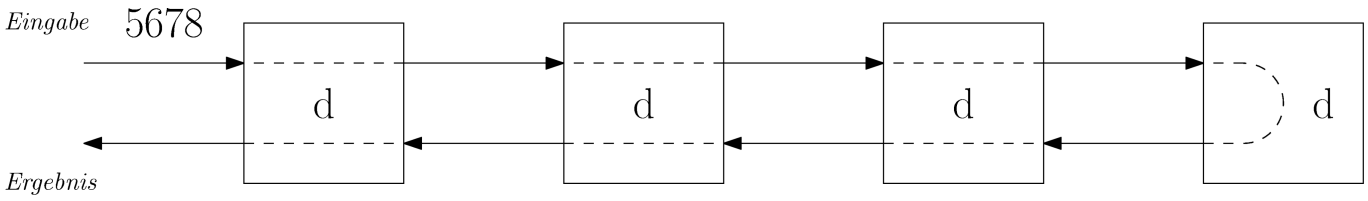
Der Datenfluss durch die Aktivierungen der Methoden kann für konkrete Beispieleingaben anhand von Diagrammen veranschaulich werden. Für die Aufrufe von `a` und `b` in `testeAlleMethoden` sehen diese wie folgt aus:

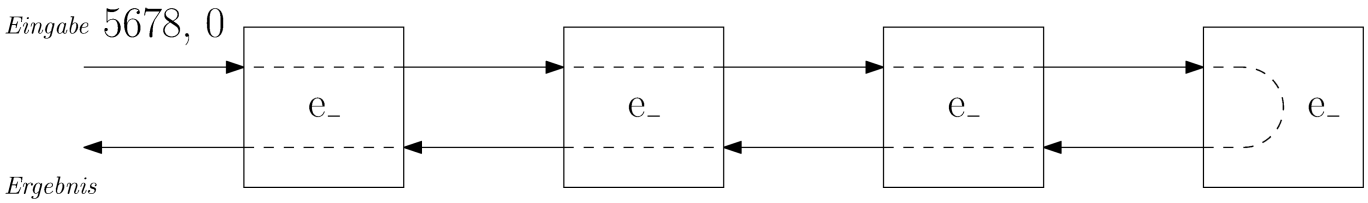


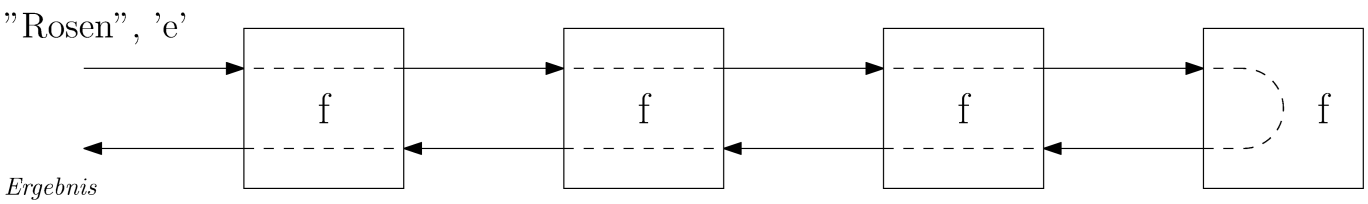


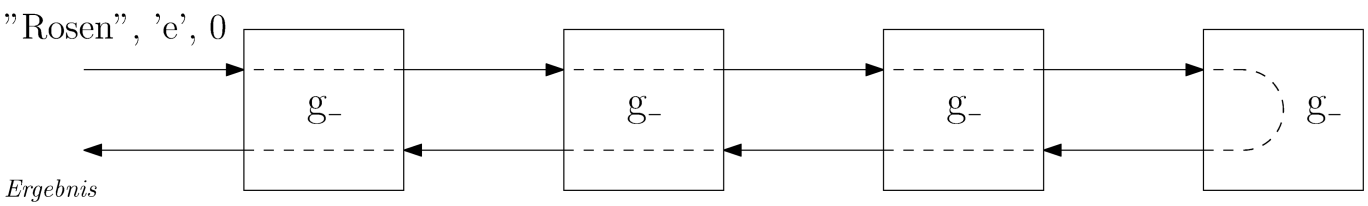
8.1.1 Vervollständigt **mindestens vier** der folgenden Diagramme für die verbleibenden fünf Methoden und erklärt eurem Betreuer **mithilfe des Debuggers** den Programmfluss:











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

Aufgabe 8.2 Eigene Rekursionen schreiben

Füllt die Rümpfe der Methoden der Klasse *Schreiben* 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 8.3 Geburtstag und Sternenhimmel

3.1.1 Weil es so viel Spaß gemacht hat, bereitet Karel die nächste Party mit Lampions vor. Allerdings ist der Fußboden von seiner feuchtfröhlichen Geburtstagsparty durchgeweicht; Karel muss aufpassen, nicht durch den Boden zu brechen und in die darunterliegende Etage zu fallen.

3.1.2 Karel trifft sich mit seiner großen Liebe Karoline zu einem romantischen Date auf einem zugefrorenen See und holt ihr die Sterne vom Himmel.

Aufgabe 8.4 Höhlenforschung und Fliesenstress

3.2.1 Karel ist Höhlenforscher geworden und verdient sich seinen Lebensunterhalt mit Touristenführungen. Um die Touristen nicht zu gefährden, bricht er sämtliche Stalaktiten ab und stellt sie als Stalagmiten wieder auf.

3.2.2 Karel betätigt sich mal wieder als Fliesenleger. Als Karoline das Ergebnis sieht, ist sie allerdings überhaupt nicht begeistert. Um den Hausfrieden zu sichern, bleibt Karel wohl nichts anderes übrig, als die verlegten Fliesen wieder zu entfernen, und zwar in umgekehrter Reihenfolge.
Bei dieser Aufgabe müssen alle elementaren Befehle, die beim rekursiven Abstieg ausgeführt wurden, beim rekursiven Aufstieg wieder rückgängig gemacht werden. Welche Befehle sind das?
Damit Karel während des rekursiven Aufstiegs entscheiden kann, ob er sich zurückdrehen muss oder nicht, muss es zwei rekursive Aufrufe geben, von denen aber immer nur einer gewählt wird. Auf keinen Fall solltest du beim rekursiven Aufstieg weitere Fallunterscheidungen vornehmen!