Karlsruher Institut für Technologie

Programmieren – Sommersemester 2017
Architecture-driven Requirements Engineering (ARE)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Jun.-Prof. Dr.-Ing. Anne Koziolek

Übungsblatt 5

Ausgabe: 20.06.2017 – 13:00
Abgabe: 05.07.2017 – 06:00

Allgemeine Hinweise

- Achten Sie darauf nicht zu lange Zeilen, Methoden und Dateien zu erstellen[1]
- Programmcode muss in englischer Sprache verfasst sein
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang`, `java.io` und `java.util`, es sei denn die Aufgabenstellung erlaubt ausdrücklich weitere Pakete[1]
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]
- Halten Sie alle Whitespace-Regeln ein[1]
- Halten Sie die Regeln zu Variablen-, Methoden und Paketbenennung ein und wählen Sie aussagekräftige Namen[1]
- Halten Sie die Regeln zu Javadoc-Dokumentation ein[1]
- Nutzen Sie nicht das default-Package[1]
- Halten Sie auch alle anderen Checkstyle-Regeln ein

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 26. Juni um 13:00 Uhr**, freigeschaltet.

- Geben Sie die Java-Klassen zu Aufgabe A als `*.java`-Dateien ab.

Wichtiger Hinweis

System.exit darf in diesem Übungsblatt nicht verwendet werden!

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

Terminal-Klasse

Laden Sie für **diese Aufgabe** die `Terminal`-Klasse[a] von unserer Homepage herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in` oder ähnliche Funktionalität, wie `java.util.Scanner`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die `Terminal`-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`.
Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,` beginnen.
Laden Sie die Terminal-Klasse niemals zusammen mit Ihrer Abgabe hoch.

[a] https://sdqweb.ipd.kit.edu/lehre/SS17-Programmieren/Terminal.java

Fehlerbehandlung

Ihre Programme sollen auf ungültige Benutzereingaben mit einer aussagekräftigen Fehlermeldung reagieren. Aus technischen Gründen muss eine Fehlermeldung unbedingt mit `Error,` beginnen. Eine Fehlermeldung führt nicht dazu, dass das Programm beendet wird; es sei denn, die nachfolgende Aufgabenstellung verlangt dies ausdrücklich. Achten Sie insbesondere auch darauf, dass unbehandelte `RuntimeExceptions`, bzw. Subklassen davon—sogenannte *Unchecked Exceptions*—nicht zum Abbruch des Programms führen.

Checkstyle

Denken Sie daran, den Checkstyle-Regelsatz von unserer Homepage zu beziehen. Planen Sie, wie immer, ausreichend Zeit für die Abgabe ein, sollte der Praktomat Ihre Abgabe wegen einer Regelverletzung ablehnen.

Öffentliche Tests

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes nötig ist. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

Objektorientierte Modellierung

Achten Sie darauf, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung, als auch Funktionalität bewertet werden.

Empfehlungssystem (20 Punkte)

In dieser Aufgabe programmieren Sie ein *Produkt-Empfehlungssystem*, das in keinem größeren Online-Shop fehlen darf. Das Empfehlungssystem erlaubt seinen Benutzern ein (für sie interessantes) *Referenzprodukt* zu benennen, wovon ausgehend verwandte Produkte ermittelt werden. Diese werden dem Benutzer als *Produktempfehlungen* ausgegeben.

Die Abbildung von einem Referenzprodukt auf passende Produktempfehlungen wird durch *Empfehlungsstrategien* bestimmt. Der Benutzer kann aus verschiedenen Empfehlungsstrategien auswählen, um das Empfehlungssystem an seine Bedürfnisse anzupassen.

Ein besonderes Merkmal des Empfehlungssystem ist die Verwaltung des Datenbestandes in Form eines *Graphen*.

Der Datenbestand beinhaltet *Produkte* und *Kategorien*, zwischen denen bestimmte *Beziehungen* aufgebaut werden können. Eine Beziehung zwischen zwei Produkten p_1 und p_2 könnte beispielsweise ausdrücken, dass p_2 Produktnachfolger von p_1 ist. Interessiert sich der Benutzer für p_1, erscheint es lohnenswert, p_2 als Produktempfehlung auszugeben.

A Datenbestand

Der Datenbestand ist als *Graph* organisiert. Produkte und Kategorien werden durch **Knoten** modelliert. Beziehungen zwischen Produkten und Kategorien werden als **Kanten** modelliert.

A.1 Produkte und Kategorien

Knoten im Graphen repräsentieren Produkte und Kategorien. Produkte und Kategorien besitzen jeweils einen *Namen*. Produkte tragen außerdem eine eindeutige *Identifikationsnummer*. Gültige Produkt- bzw. Kategorienamen beschreibt der reguläre Ausdruck `[a-zA-Z0-9]+`. Gültige Identifikationsnummern sind positive Integer-Zahlen inklusive 0.

Weiterhin sollen im Rahmen der Aufgabe auch Produkt- und Kategorienamen eindeutig sein. Das bedeutet, wenn zwei Knoten n_1 und n_2 den gleichen Namen tragen (oder Produktknoten dieselbe Identifikationsnummer), muss gelten: $n_1 = n_2$.

Knotennamen, die sich nur durch Groß-/Kleinschreibung unterscheiden, werden als identisch betrachtet. Bei zwei Knoten mit dem Namen „libreoffice“ und „LibreOffice“ handelt es sich also um denselben Knoten.

A.2 Beziehungen

Kanten im Graphen setzen Produkte und Kategorien zueinander in Beziehung. Dafür existieren folgende *Beziehungstypen*.

Beachten Sie im Folgenden, dass jede Art von Beziehung eine *Umkehrbeziehung* besitzt.

- **„contains“-Beziehung**: n_1 *contains* n_2 drückt aus, dass das Produkt oder die Kategorie n_1 das Produkt oder die Kategorie n_2 enthält. Dabei gilt, dass wenn n_1 ein Produkt repräsentiert, n_2 nur ein Produkt sein darf. Ist n_1 eine Kategorie, so darf n_2 entweder ein Produkt oder eine Kategorie sein. Beispiel: `software contains operatingsystem`. Sind beide Knoten vom Typ Kategorie, so drückt diese Beziehung eine Spezialisierung der Oberkategorie n_1 durch die Unterkategorie n_2 aus. (Umkehrbeziehung: *contained-in*)
- **„contained-in“-Beziehung**: n_1 *contained-in* n_2 drückt aus, dass Produkt oder Kategorie n_1 in Produkt oder Kategorie n_2 enthalten ist. Beispiel: `operatingsystem contained-in software`. Dabei gilt, dass wenn n_2 ein Produkt repräsentiert, n_1 nur ein Produkt sein darf. Ist n_2 eine Kategorie, so darf n_1 entweder ein Produkt oder eine Kategorie sein.. Sind beide Knoten vom Typ Kategorie, so drückt diese Beziehung eine Spezialisierung der Oberkategorie n_2 durch die Unterkategorie n_1 aus. (Umkehrbeziehung: *contains*)
- **„successor-of“-Beziehung**: n_1 *successor-of* n_2 drückt aus, dass Produkt n_1 das (direkte) Nachfolgeprodukt von n_2 ist. Beispiel: `centos7 successor-of centos6`. Dabei sind n_1 und n_2 stets Produkte. (Umkehrbeziehung: *predecessor-of*)
- **„predecessor-of“-Beziehung**: n_1 *predecessor-of* n_2 drückt aus, dass Produkt n_1 das (direkte) Vorgängerprodukt von n_2 ist. Beispiel: `centos6 predecessor-of centos7`. Dabei sind n_1 und n_2 stets Produkte. (Umkehrbeziehung: *successor-of*)

Wird dem Graphen mittels einer Kante $e = (n_1, n_2, b)$ die Beziehung b von Knoten n_1 zu Knoten n_2 hinzugefügt, **soll zusätzlich die Umkehrbeziehung $\bar{b}$ mittels der Kante $e = (n_2, n_1, \bar{b})$ hergestellt werden**, sofern diese nicht schon vorhanden sind.

Denn, redundante Kanten sind zu vermeiden: Für zwei Kanten $e_1 = (n_{1_1}, n_{1_2}, b_1)$ und $e_2 = (n_{2_1}, n_{2_2}, b_2)$ muss $e_1 = e_2$ gelten, genau dann wenn $n_{1_1} = n_{2_1}$ und $n_{1_2} = n_{2_2}$ und $b_1 = b_2$.

Weiterhin kann ein Knoten nicht zu sich selbst in Beziehung gesetzt werden: Für jede Kante $e = (n_1, n_2, b)$ gilt $n_1 \neq n_2$.

B Empfehlungsstrategien

Eine Empfehlungsstrategie berechnet ausgehend von einem Referenzprodukt eine Menge von verwandten Produkten. Dazu operiert jede Empfehlungsstrategie auf der Datenbasis und macht sich inbesondere die durch Beziehungen gegebenen semantischen Informationen zunutze.

Für alle im Folgenden eingeführten Empfehlungsstrategien gilt, dass das Referenzprodukt niemals Teil der empfohlenen Produktmenge ist und gegebenenfalls entfernt werden muss bevor die Empfehlungen zurückgeliefert werden. Im Folgenden werden drei *einfache Empfehlungsstrategien* spezifiziert:

Weiterhin sollen ausschließlich Produkte empfohlen werden, niemals Kategorien.

- **Strategie „Geschwisterprodukte“ (S1)** liefert für ein bestimmtes Referenzprodukt p_r alle Produkte, die zusammen mit p_r in einer **direkten** „contained-in“-Beziehung zu einer gemeinsamen Oberkategorie stehen. Nehmen wir am Beispiel der Abbildung 1 an, dass „centos6“ das Referenzprodukt ist. Diese Strategie empfiehlt dann die Produkte „centos5“ und „centos7“.
- **Strategie „Nachfolgeprodukte“ (S2)** liefert für ein bestimmtes Referenzprodukt p_r **alle direkten und indirekten Nachfolgeprodukte** von p_r durch Verfolgung der „predecessor-of“-Beziehung beginnend mit p_r. Nehmen wir am Beispiel der Abbildung 1 an, dass „centos5“ das Referenzprodukt ist. Diese Strategie empfiehlt dann die Produkte „centos6“ und „centos7“. Beachten Sie, dass ein Produkt mehrere direkte Nachfolger haben kann. Ein Beispiel dafür ist die Office-Suite Open-Office, aus der die Produkte Apache OpenOffice sowie LibreOffice als zwei unabhängige Nachfolger hervorgingen (nicht in Abb. 1 dargestellt).
- **Strategie „Vorgängerprodukte“ (S3)** liefert für ein bestimmtes Referenzprodukt p_r **alle direkten und indirekten Vorgängerprodukte** von p_r durch Verfolgung der „sucessor-of“-Beziehung beginnend mit p_r. Nehmen wir am Beispiel der Abbildung 1 an, dass „centos7“ das Referenzprodukt ist. Diese Strategie empfiehlt dann die Produkte „centos6“ und „centos5“. Beachten Sie, dass ein Produkt mehrere direkte Vorgänger haben kann. Um beim Beispiel Open-Office zu bleiben, etwa wenn der (unwahrscheinliche) Fall eintreten würde, dass die Projekte Apache OpenOffice und LibreOffice ihre Kräfte wieder in einem gemeinsam Produkt bündeln. (nicht in Abb. 1 dargestellt).

C Kommandozeilenparameter

Beim Start erwartet Ihr Programm als erstes (und einziges) Kommandozeilenargument einen Pfad auf eine Textdatei. Diese Textdatei beinhaltet die im Datenbestand zu speichernden Produkte, Kategorien sowie deren Beziehungen.

C.1 Aufbau der Datenbestand-Datei

Die Datenbestand-Datei besteht aus einer oder mehreren Zeilen. Jede Zeile stellt durch ein Prädikat eine Beziehung zwischen einem Subjekt und einem Objekt her. Das Format einer solchen Zeile ist durch folgende BNF-Grammatik[2] gegeben. Terminalsymbole sind im `Typewriter`-Stil gesetzt. Abweichend von der BNF sind die Nichtterminalsymbole *productid*, *productname* sowie *categoryname* jeweils durch einen regulären Ausdruck spezifiziert, der die Menge der gültigen Terminalsymbole ausdrückt.

[2] Backus-Naur-Form, siehe z.B. `http://de.wikipedia.org/wiki/Backus-Naur-Form`

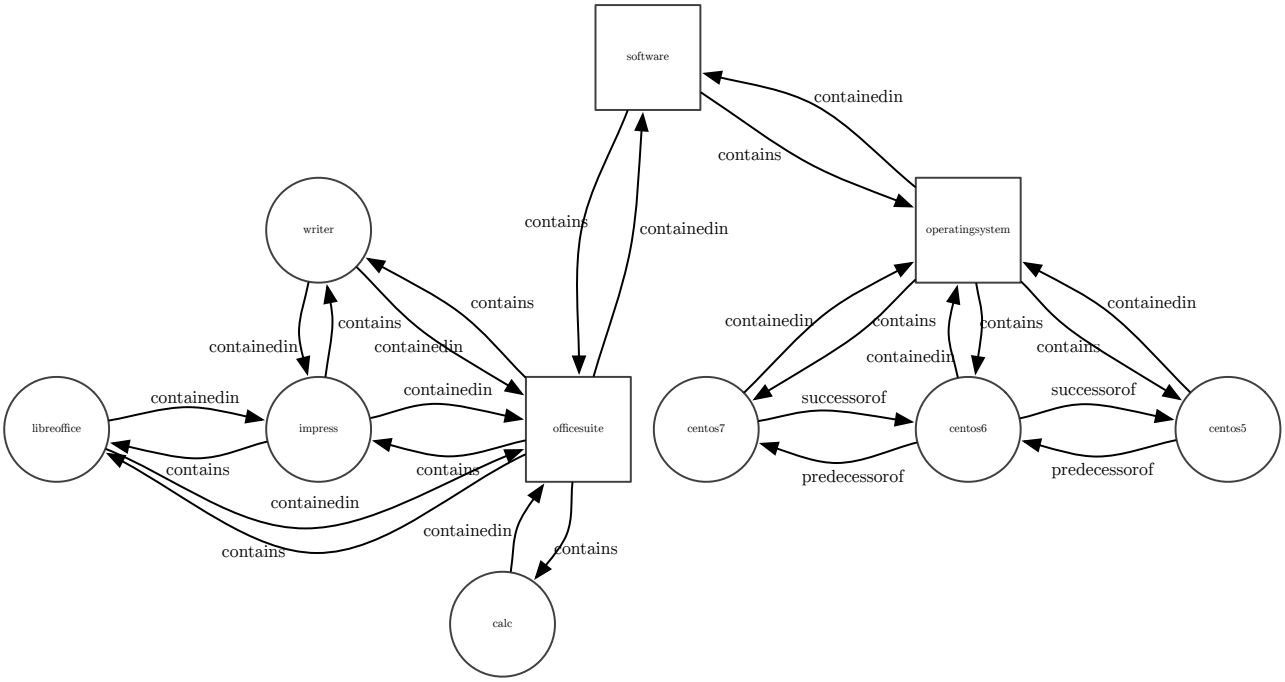


Abbildung 1: Graph eines Beispiel-Datenbestandes

BNF-Grammatik einer Zeile der Datenbestand-Datei

line ::= *subject predicate object*
subject ::= *product* | *categoryname*
object ::= *product* | *categoryname*
predicate ::= `contains` | `contained-in` | `successor-of` | `predecessor-of`
product ::= *productname* `(id=` *productid* `)`
productid ::= `[0-9]+`
productname `[a-zA-Z0-9]+`
categoryname ::= `[a-zA-Z0-9]+`

Ergänzend zu obiger Grammatik sollen unnötige Leerzeichen ignoriert werden anstatt eine Fehlermeldung auslösen, und zwar noch folgenden Regeln.

Innerhalb der Ableitungsregel *line* sind beide dargestellten Leerzeichen erforderlich; zusätzliche Leerzeichen dürfen vor/nach *subject*, *predicate*, *object* stehen. Innerhalb der Ableitungsregel *product* ist keines der dargestellen Leerzeichen erforderlich; zusätzliche Leerzeichen düfen vor/nach *productname*, `(`, `id`, `=`, *productid*, `)` stehen. Alle anderen Ableitungsregeln folgen strikt der entsprechenden BNF-Spezifikation.

So soll beispielsweise eine Zeile der folgenden Form akzeptiert werden:

```
centos7 ( id = 107 ) contained-in operatingsystem
```

Folgende Bestandteile dürfen jedoch absolut keine Leerzeichen enthalten: *predicate*, *productid*, *productname*, *categoryname*.

Denken Sie beim Aufbau des Graphen daran, Umkehrbeziehungen herzustellen (vgl. Abschn. A.2), falls diese nicht bereits in der Datenbestand-Datei beschrieben sind.

Tritt beim Einlesen der Datei ein Fehler auf, sei es ein Ein-/Ausgabefehler beim Lesen der Datei, ein Syntaxfehler,

oder ein semantischer Fehler, so wird eine Fehlermeldung ausgegeben und das Programm beendet sich. Ein Syntaxfehler besteht, wenn die Datei nicht gemäß obenstehender Spezifikation geformt ist. Ein semantischer Fehler besteht, wenn beispielsweise eine ungültige Beziehung aufgebaut wird, etwa zwei Kategorien, die mittels der `successor-of`-Beziehung verbunden werden sollen.

C.2 Beispiel einer Datenbestand-Datei

Folgendes Beispiel präsentiert eine wohlgeformte Datenbestand-Datei zum Aufbau des in Abbildung 1 dargestellten Graphen:

```
CentOS5 ( id= 105) contained-in operatingSystem
centOS6 ( id = 106) contained-in OperatingSystem
operatingSystem contains centos7 ( id = 107 )
operatingsystem contained-in Software
CentOS7 (id=107) successor-of centos6(id=106)
CentOS5 (id=105) predecessor-of centos6(id=106)
writer (id=201) contained-in officesuite
writer (id=201) contained-in impress (id=203)
calc (id=202) contained-in officesuite
impress (id=203) contained-in officesuite
officesuite contained-in software
LibreOffice (id=200) contained-in officesuite
(...)
```

Das Auslassungssymbol `(...)` steht stellvertretend für Ausgabezeilen, die aus Platzgründen entfernt wurden.

Der in Abbildung 1 abgebildete Graph kann mit unterschiedlichen Eingaben erzeugt werden. Die obige Eingabe ist somit nur eine Beispieleingabe für den Aufbau dieses Graphen.

D Interaktive Benutzerschnittstelle

Nach dem Start nimmt Ihr Programm über die Konsole mittels `Terminal.readLine()` fünf Arten von Befehlen entgegen. Nach Abarbeitung eines Befehls wartet das Programm auf weitere Befehle, bis das Programm irgendwann durch `quit` beendet wird.

Alle Befehle (außer `quit`) operieren auf dem Graphen, der zuvor durch Einlesen der Datenbestand-Datei aufgebaut wurde.

D.1 Der `nodes`-Befehl

Der `nodes`-Befehl gibt eine Liste aller Knoten aus.

D.1.1 Ausgabeformat

Die Knoten werden komma-separiert in einer einzigen Zeile ausgegeben. Ein *Produktknoten* wird dagestellt durch dessen Namen in Kleinschreibung, gefolgt von einem Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Ein *Kategorieknoten* wird dagestellt durch dessen Namen in Kleinschreibung.

Die Knoten werden aufsteigend sortiert nach Produkt- bzw. Kategorienamen ausgegeben.

D.1.2 Beispiel

```
nodes
calc:202,centos5:105,centos6:106,centos7:107,impress:203,libreoffice:200,officesuite,
    operatingsystem,software,writer:201
```

Beachten Sie, dass in obigem Beispiel Zeile 2 und 3 tatsächlich zur selben Zeile gehören und hier nur aus Platzgründen umgebrochen sind.

D.2 Der `edges`-Befehl

Der `edges`-Befehl gibt eine Liste aller Kanten aus.

D.2.1 Ausgabeformat

Die Kanten werden zeilenweise ausgegeben, wobei jede Kante in einer separaten Zeile steht. Jede dieser gerichteten Kanten besitzt im Rahmen dieser Aufgabe drei Bestandteile: Quellknoten, Beziehungstyp, Zielknoten. Quell- und Zielknoten werden jeweils dagestellt durch deren Namen in Kleinschreibung, gefolgt von einem Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Der Beziehungstyp wird eingeleitet durch die Zeichen „`-[`“, gefolgt von dem Namen des Beziehungstyps in Kleinschreibung, gefolgt von den Zeichen „`]->`“.

Die Reihenfolge der ausgegebenen Kanten ist beliebig.

D.2.2 Beispiel

```
edges
calc:202-[contained-in]->officesuite
centos5:105-[contained-in]->operatingsystem
centos5:105-[predecessor-of]->centos6:106
centos6:106-[contained-in]->operatingsystem
centos6:106-[predecessor-of]->centos7:107
centos6:106-[successor-of]->centos5:105
centos7:107-[contained-in]->operatingsystem
centos7:107-[successor-of]->centos6:106
impress:203-[contained-in]->officesuite
(...)
```

Das Auslassungssymbol `(...)` steht stellvertretend für Ausgabezeilen, die aus Platzgründen entfernt wurden.

D.3 Der `recommend`-Befehl

Mithilfe des `recommend`-Befehls kann sich der Benutzer Produkte empfehlen lassen. Dabei wird das jeweilige Referenzprodukt durch dessen Produktidentifikationsnummer repräsentiert.

D.3.1 Eingabeformat

Ein gültiger `recommend`-Befehl ist wie folgt aufgebaut. Wie bereits zuvor werden reguläre Ausdrücke als BNF-Erweiterung eingesetzt und Terminalsymbole sind im `Typewriter`-Stil gesetzt.

BNF-Grammatik des recommend-Befehls

command ::= `recommend` *term*
term ::= *final*
final ::= *strategy productid*
strategy ::= `S1` | `S2` | `S3`
productid ::= `[0-9]+`

Ergänzend zu obiger Grammatik sollen unnötige Leerzeichen (nur) innerhalb der Nichtterminale *term* sowie *final* ignoriert werden anstatt eine Fehlermeldung auszulösen. So soll beispielsweise eine Eingabe der folgenden Form akzeptiert werden:

```
recommend    S1 105
```

D.3.2 Ausgabeformat

Die empfohlenden Produkte werden komma-separiert in einer einzigen Zeile ausgegeben. Jedes Produkt wird dagestellt durch dessen Namen in Kleinschreibung, gefolgt von einem Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Die Produkte werden aufsteigend sortiert nach Produktnamen ausgegeben. Ist die Menge der empfohlenen Produkte leer, so wird eine leere Zeile ausgegeben.

D.3.3 Beispiel

Im Folgenden ist eine Sequenz gültiger Ein- und Ausgaben dargestellt.

```
recommend S1 105
centos6:106,centos7:107
recommend S3 107
centos5:105,centos6:106
recommend S1 201
calc:202,impress:203,libreoffice:200
```

D.4 Der `export`-Befehl

Der `export`-Befehl gibt den Graphen des eingelesenen Datenbestands in DOT-Notation auf die Konsole aus.

Details zur DOT-Notation finden Sie in Abschnitt E. Nutzen Sie für den `export`-Befehl bitte ausschließlich die Sprachkonstrukte aus Abschnitt E. Andernfalls könnten die automatischen Tests fehlschlagen.

D.4.1 Ausgabeformat

Entsprechend der DOT-Notation beginnt die Ausgabe mit der Zeile „`digraph {`“ und endet mit der Zeile „`}`“. Dazwischen geben Sie für jede Kante des Graphen genau eine Zeile in DOT-Notation aus. Kanten tragen als Label ihren jeweiligen Beziehungstyp, wobei Bindestriche entfernt werden. Gültige Kantenlabel sind somit beispielsweise `containedin` sowie `successorof`. Produktknoten mit eingehenden und/oder ausgehenden Kante müssen nicht in einer separaten Zeile beschrieben werden—diese werden indirekt durch die Kanten beschrieben. Kategorieknoten hingegen sollen als Rechteckt dargestellt werden. Dazu ist es nötig, jeden Kategorieknoten als separate Zeile auszugeben. Die Reihenfolge der Zeilen spielt keine Rolle, abgesehen von der einleitenden und abschließenden Zeile. Produktknoten, ebenso wie Kategorieknoten, sollen durch deren *Namen* dargestellt werden. Eine Zeile darf mit Leerzeichen begonnen werden, solange es sich nicht um die einleitende oder abschließende Zeile handelt.

D.4.2 Beispiel

```
export
digraph {
  centos7 -> operatingsystem [label=containedin]
  centos6 -> operatingsystem [label=containedin]
  centos5 -> operatingsystem [label=containedin]
  calc -> officesuite [label=containedin]
  officesuite -> impress [label=contains]
  software -> operatingsystem [label=contains]
  impress -> writer [label=contains]
  officesuite -> libreoffice [label=contains]
  impress -> officesuite [label=containedin]
  software -> officesuite [label=contains]
  centos6 -> centos7 [label=predecessorof]
  officesuite -> calc [label=contains]
  operatingsystem -> centos5 [label=contains]
  centos5 -> centos6 [label=predecessorof]
  operatingsystem -> centos6 [label=contains]
  officesuite -> writer [label=contains]
  libreoffice -> officesuite [label=containedin]
  operatingsystem -> centos7 [label=contains]
  writer -> officesuite [label=containedin]
  centos7 -> centos6 [label=successorof]
  officesuite -> software [label=containedin]
  operatingsystem -> software [label=containedin]
  writer -> impress [label=containedin]
  centos6 -> centos5 [label=successorof]
  software [shape=box]
  officesuite [shape=box]
  operatingsystem [shape=box]
}
```

D.5 Der `quit`-Befehl

Dieser Befehl beendet das Programm. Dabei findet keine Konsolenausgabe statt.

E DOT-Notation

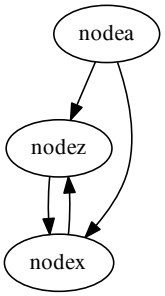
Die DOT-Notation wird zur Beschreibung von Graphen verwendet. Ein besonderes Merkmal dieser Notation ist, dass sie sowohl von Menschen, als auch von Maschinen lesbar ist[3].

E.1 Gerichteter Graph

Die Notation eines gerichteten Graphen beginnt mit dem Schlüsselwort `digraph`. Die Beschreibung von Knoten und Kanten erfolgt innerhalb von zwei geschweiften Klammern. Die Namen der Knoten sind beliebig wählbar. Eine gerichtete Kante wird durch „->“ definiert. Im Folgenden ist ein Graph in DOT-Notation, sowie seine entsprechende visuelle Darstellung abgebildet.

[3] Siehe hierzu `http://www.graphviz.org/content/dot-language`, sowie `http://en.wikipedia.org/wiki/DOT_(graph_description_language)`)

```
digraph {
  nodea -> nodez
  nodea -> nodex
  nodez -> nodex
  nodex -> nodez
}
```



E.2 Weitere Merkmale der DOT-Notation

Um ein Knoten als ein Rechteck darstellen zu können, benutzen Sie den Namen des Knotens und das Schlüsselwort `[shape=box]`.

Um Kanten mit einer Beschriftung zu versehen, nutzen Sie `[label=edgename]`.

Folgendes Beispiel illustriert diese Fälle:

```
digraph {
  nodez [shape=box]
  nodea -> nodez
  nodea -> nodex [label=edgeax]
  nodez -> nodex
  nodex -> nodez
}
```