Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Architecture-driven Requirements Engineering (ARE)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
programmieren-vorlesung@ipd.kit.edu

Programmieren – Sommersemester 2020

Übungsblatt 3 Version 1.2	20 Punkte	Ausgabe:	25.05.2020, ca. 13:00 Uhr
		Praktomat:	01.06.2020, 13:00 Uhr
		Abgabefrist:	09.06.2020, 06:00 Uhr

Abgabehinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Der Praktomat wird Ihre Abgabe zurückweisen, falls eine der nachfolgenden Regeln verletzt ist. Eine zurückgewiesene Abgabe wird automatisch mit null Punkten bewertet. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie keine Elemente der Java-Bibliotheken, ausgenommen Elemente der Pakete `java.lang`, `java.util` und `java.util.regex`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()` und `Runtime.exit()` dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln an.

Bearbeitungshinweise

Diese Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe, jedoch wird der Praktomat Ihre Abgabe **nicht** zurückweisen, falls eine der nachfolgenden Regeln verletzt ist.

- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Plagiat

Es werden nur selbstständig angefertigt Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden und alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Ebenso stellt die Weitergabe einer Lösung oder von Teilen davon eine Störung des ordnungsgemäßen Ablaufs der Erfolgskontrolle dar. Dieser Ordnungsverstoß kann ebenfalls zum Ausschluss der Erfolgskontrolle führen. Für weitere Ausführungen sei auf die Einverständniserklärung (Disclaimer) verwiesen.

Checkstyle

Der Praktomat überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen der Praktomat die Abgabe zurückweist, da diese Regel verpflichtend einzuhalten ist. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki im ILIAS beschreibt, wie Checkstyle verwendet wird.

>_ Terminal-Klasse

Laden Sie für diese Aufgabe die Terminal-Klasse 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`. 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`. Laden Sie die Terminal-Klasse niemals zusammen mit Ihrer Abgabe hoch.

Interaktive Benutzerschnittstelle

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Setzen Sie nur die in der Aufgabenstellung angegebenen Informationen um. Geben Sie auch keine zusätzlichen Informationen aus. Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error ,` beginnen und darf keine Zeilenumbrüche enthalten.

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 1. Juni 2020 um 13:00 Uhr**, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Praktomat bei der richtigen Aufgabe vor Ablauf der Abgabefrist hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären.

- Geben Sie Ihre Klassen zu Aufgabe A als `*.java`-Dateien ab.
- Geben Sie Ihre Klassen zu Aufgabe B als `*.java`-Dateien ab.
- Geben Sie Ihre Klassen zu Aufgabe C als `*.java`-Dateien ab.
- Geben Sie Ihre Klasse zu Aufgabe D als `*.java`-Datei ab.

Lernziele

Der Fokus dieses Übungsblattes liegt in den nachfolgenden Lernzielen aus der Vorlesung, inklusive der Übung. Beachten Sie, dass alle anderen bisherigen Lernziele ebenso gelten. Berücksichtigen Sie daher generell den gesamten bisherigen Inhalt aus Vorlesung und Übung.

- Konvertierung
  - Sie wissen, welche Konvertierungen Java automatisch vornimmt
  - Sie können Konvertierungen explizit über den Cast-Operator vornehmen
  - Sie können eine Methode zur Konvertierung von Objekten Ihrer eigenen Klassen in Strings angeben
- Datenkapselung & Sichtbarkeit
  - Sie können Attribute und Methoden Ihrer Klassen nach außen verbergen
  - Sie kennen die Bedeutung der einzelnen Zugriffsrechte in Java
  - Sie kennen den Gültigkeitsbereich und den Sichtbarkeitsbereich von Variablen sowie ihre Lebensdauer

- Listen & Abstrakte Datentypen
  - Sie kennen den Aufbau von einfach und mehrfach verketteten Listen
  - Sie können Daten in Listen ablegen, verändern, und löschen
  - Sie können Iteratoren verwenden, um über Listen zu iterieren
  - Sie können die Vor- und Nachteile von Listen gegenüber Arrays benennen
  - Sie kennen die Bestandteile von abstrakten Datentypen
  - Sie können beschreiben, inwiefern eine Liste ein abstrakter Datentyp ist

Aufgabe A: Wiedergabeliste (6 Punkte)

In dieser Aufgabe soll eine Wiedergabeliste für einen Radiosender mit beliebigen Prioritäten implementiert werden. Eine Wiedergabeliste soll dabei per Vorrangwarteschlange modelliert werden, welche die aktuell laufenden und geplanten Lieder des Radiosenders, sortiert nach ihrer Priorität, beinhaltet.

Dabei wird ein Lied durch eine Kennung identifiziert und hat als weitere Attribute neben seiner Priorität auch einen Interpreten, einen Titel und eine Dauer. Ein Lied kann der Wiedergabeliste mehrfach hinzugefügt werden, d.h. die Wiedergabeliste kann Lieder mit identischen Kennungen und Attributen enthalten. Die Wiedergabeliste kann jedoch keine Lieder mit identischen Kennungen und unterschiedlichen Attributen enthalten. Die Kennung für ein Lied kann nach dem Entfernen neu vergeben werden, ebenso kann die Wiedergabeliste auch Lieder mit identischen Attributen, aber unterschiedlichen Kennungen enthalten.

Eine Vorrangwarteschlange (Prioritätenliste, eng. *priority queue*) ist eine spezielle abstrakte Datenstruktur, genauer gesagt eine erweiterte Form einer Warteschlange. Den Elementen, welche in die Warteschlange eingereiht werden, wird ein Schlüssel (in unserem Fall Prioritäten genannt) mitgegeben, der die Reihenfolge der Abarbeitung der Elemente bestimmt[1].

Ein Moderator kann diese Liste geplanter Lieder mithilfe von Befehlen aggregieren und den Zeitpunkt der Ausstrahlung dynamisch bestimmen, indem er die Priorität der Lieder angibt. Hinzugefügte Lieder werden in eine Vorrangwarteschlange einsortiert und entsprechend abgespielt. Es ist auch möglich, die Wiedergabe für eine gegebene Sekundenzahl zu simulieren. In diesem Fall wird die Zeit des aktuellen wiedergegebenen Liedes heruntergezählt. Wenn ein Lied während einer Simulation nicht vollständig abgespielt wurde, bleibt die verbleibende Zeit des Liedes für die nächste Simulation erhalten. Wenn das Lied beendet ist, wird auch die Zeit des nächsten Liedes heruntergezählt, und so weiter.

A.1 Interaktive Benutzerschnittstelle

Nach dem Start nimmt das Programm über die Konsole Befehle unter Verwendung der Methode `readLine` der Klasse `Terminal` entgegen. Nach der Verarbeitung eines Befehls wartet das Programm auf weitere Eingaben, bis es durch Eingabe des `quit`-Befehls beendet wird. Die Eingabe und die Ausführung der Befehle dürfen nicht gegen die definierten Spezifikationen verstoßen. Bei einem Verstoß muss immer eine aussagekräftige Fehlermeldung ausgegeben, anschließend wartet das Programm regulär auf weitere Eingaben. Beachten Sie, dass nicht bei jeder erfolgreichen Interaktion automatische eine Ausgabe erfolgen muss.

[1] `https://de.wikipedia.org/wiki/Vorrangwarteschlange`

A.1.1 Terminalsymbole

<id> Ganzzahlige Kennung im offenen Intervall von $(0, 10^5)$. Wird immer mit Nullen auf der linken Seite zu insgesamt fünf Zeichen für die Ausgabe aufgefüllt.

<artist> <title> Der Interpret oder Titel eines Liedes, jeweils besteht aus einem oder mehreren Buchstaben des deutschen Alphabets[2].

<length> <time> Ganzzahlige Zeiteinheit in Sekunden im offenen Intervall von $(0, 2^{31})$.

<priority> Ganzzahlige Priorität im abgeschlossenen Intervall von $[0, 5]$, wobei 0 für die höchste und 5 für die niedrigste Priorität steht.

<amount> Positive ganze Zahl, welche die Anzahl der entfernten Lieder angibt.

A.1.2 Befehle

add <id>:<artist>:<title>:<length>:<priority> Fügt der Wiedergabeliste ein neues Lied mit den angegebenen Attributen hinzu. Basierend auf der gegebenen Priorität wird dieses Lied am Ende des entsprechenden Prioritätsbereichs in die Wiedergabeliste eingereiht. Das gleiche Lied kann mehrfach hinzugefügt werden. Ein neues Lied wird nach allen bereits gestarteten Liedern hinzugefügt. Folglich wird durch das Hinzufügen eines Liedes mit einer höheren Priorität bereits laufende Lieder nicht unterbrochen, sondern dieses neue Lied dahinter eingereiht.

remove <id> Entfernt alle Lieder mit einer gegebenen Kennung aus der Wiedergabeliste. Wenn es keine Lieder mit der Kennung gibt, wird kein Lied entfernt und der Fall als erfolgreich betrachtet. Wenn ein oder mehrere Lieder entfernt wurden, wird `Removed <amount> songs.` ausgegeben.

play <length> Simuliert die Wiedergabe der Wiedergabeliste für die Zeitspanne der angegebenen Sekunden. Lieder, welche in der angegebenen Zeitspanne fertiggestellt werden, müssen aus der Wiedergabeliste entfernt werden. Wenn die Wiedergabeliste vor der angegebenen Zeit leer ist, wird der Fall als erfolgreich betrachtet.

skip Überspringt das nächste abzuspielende Lied und entfernt es aus der Wiedergabeliste. Wenn sich kein Lied in der Wiedergabeliste befindet, gilt der Fall als erfolgreich.

[2] `https://de.wikipedia.org/wiki/Deutsches_Alphabet`

`peek` Zeigt das nächste Lied an, das nach der Priorität abgespielt wird. Wenn kein Lied vorhanden ist, wird nichts ausgegeben. Im Erfolgsfall wird `<id>:<artist>:<title>:<length>:<time>` ausgegeben. Dabei steht `<time>` für die verbleibende Zeit für die Simulation eines Liedes. `<time>` ist für noch nicht laufende Lieder identisch mit `<length>`.

`list` Zeigt die vollständige Wiedergabeliste, sortiert nach Priorität, an. Bereits laufende Lieder sind hierbei ebenfalls enthalten. Wenn sich kein Lied in der Wiedergabeliste befindet, wird nichts ausgegeben. Befinden sich ein oder mehrere Lieder in der Wiedergabeliste, wird beginnend mit der höchsten Priorität ein Lied pro Zeile in der Form `<id>:<artist>:<title>:<length>` ausgegeben.

`history` Zeigt die Liste aller zuvor vollständig simulierten Lieder an. Wenn ein Lied nicht zuvor simuliert abgespielt wurde, wird nichts ausgegeben. Wenn ein oder mehrere Lieder zuvor vollständig simuliert abgespielt wurden, werden diese Lieder in der Reihenfolge ausgegeben, in der sie gespielt wurden, d.h. das zuerst gespielte Lied wird auch zuerst (oben) ausgegeben. Dabei wird pro Zeile ein Lied in der Form `<id>:<artist>:<title>:<length>` ausgegeben.

`quit` Der Befehl beendet das Programm vollständig.

A.2 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Die Eingabezeilen werden mit dem Größer-als-Zeichen gefolgt von einem Leerzeichen eingeleitet, diese beiden Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen nur der Unterscheidung zwischen Ein- und Ausgabezeilen.

Beispielinteraktion

```
1  > add 2:Alice:Spam:200:1
2  > add 40:Bob:Fubar:300:2
3  > peek
4  00002:Alice:Spam:200:200
5  > list
6  00002:Alice:Spam:200
7  00040:Bob:Fubar:300
8  > add 2:Alice:Spam:200:1
9  > add 13:Bob:FizzBuzz:400:0
10 > add 2:Alice:Spam:200:1
11 > list
12 00013:Bob:FizzBuzz:400
13 00002:Alice:Spam:200
14 00002:Alice:Spam:200
15 00002:Alice:Spam:200
16 00040:Bob:Fubar:300
17 > remove 2
18 Removed 3 songs.
19 > list
20 00013:Bob:FizzBuzz:400
21 00040:Bob:Fubar:300
22 > add 2:Alice:Eggs:100:0
23 > list
24 00013:Bob:FizzBuzz:400
25 00002:Alice:Eggs:100
26 00040:Bob:Fubar:300
27 > history
28 > play 200
29 > peek
30 00013:Bob:FizzBuzz:400:200
31 > skip
32 > peek
33 00002:Alice:Eggs:100:100
34 > play 300
35 > peek
36 00040:Bob:Fubar:300:100
37 > history
38 00002:Alice:Eggs:100
39 > quit
```

Aufgabe B: Wörterbuch (5 Punkte)

Ziel dieser Aufgabe ist es, eine geeignete Datenstruktur zu finden und zu implementieren, welche ein Wörterbuch abspeichern und verwalten kann. Dem Programm wird als Kommandozeilenargument ein Pfad zu einer Textdatei übergeben, welche eine Liste von Wörterpaaren enthält, welche in Abschnitt B.1 beschrieben sind. Diese Liste wird eingelesen und soll in einer geeigneten Datenstruktur abgespeichert werden. Alle eingelesenen Wörter aus der „Startsprache“ sollen alphabetisch sortiert[3] gespeichert werden.

In dieser Aufgabe wird immer eine alphabetische Sortierung verwendet, welche mit der in deutschen Wörterbüchern, z.B. Duden oder Wahrig, vergleichbar ist. Hierbei basiert die Sortierung auf der Buchstabenfolge des deutschen Alphabets. Bei gleicher Buchstabenfolge werden Kleinbuchstaben vor Großbuchstaben gesetzt. Die Umlaute werden mit den Grundbuchstaben gleichgesetzt, z.B. sind `Ä` und `A` gleich. Nur wenn die Schreibweise ansonsten gleich ist, steht der Grundbuchbuchstabe an erster Stelle, z.B. `Buhl` vor `Bühl`. Der Eszett wird wie `ss` geordnet, nur wenn die Schreibweise ansonsten gleich ist, steht `ss` vor `ß`.

Nach dem Einlesen gibt es die Möglichkeit über eine interaktive Kommandozeilenschnittstelle Befehle, welche in Abschnitt B.2.2 beschrieben sind, auszuführen. Dazu gehören unter anderem das Hinzufügen oder Entfernen von Wörtern und Übersetzungen sowie das Übersetzen von Wörtern und einfachen Sätzen.

B.1 Wörterlistendatei

Ihr Programm nimmt als einziges Kommandozeilenargument einen Pfad auf eine Textdatei entgegen. Verwenden Sie die Methode `readFile` der Klasse `Terminal`, um den Inhalt dieser Datei zu lesen. Hierbei können Sie nicht davon ausgehen, dass die gegebene Eingabe der gegebenen Spezifikationen entspricht. Fehlerhafte Textdateien haben keine weiteren Auswirkungen auf das Programm, es wird lediglich eine entsprechende Fehlermeldung angezeigt. Änderungen am Wörterbuch müssen nicht in die Wörterbuchdatei zurückübertragen werden, d.h. die Textdatei darf nicht modifiziert werden.

Die Wörterlistendatei enthält eine oder mehrere Zeilen. Jede Zeile beschreibt ein Wörterpaar. Ein Wörterpaar besteht aus exakt zwei Wörtern, die durch ein Separator voneinander getrennt sind. Jedes Wort besteht aus einer Sequenz einem oder mehreren Buchstaben des deutschen Alphabets[4]. Das Separator besteht aus den drei Zeichen mit dem ASCII-Hexadezimalcodes `20 2D 20`. Links neben dem Separator steht das Wort aus der „Startsprache“ und rechts neben dem Separator steht das Wort aus der „Zielsprache“. Einem Wort aus der Startsprache können mehrere Wörter aus der Zielsprache zugewiesen werden. Eine beispielhafte Wörterlistendatei könnte so aussehen:

[3] `https://de.wikipedia.org/wiki/Alphabetische_Sortierung`
[4] `https://de.wikipedia.org/wiki/Deutsches_Alphabet`

```
Der - The
Die - The
Das - The
Haus - House
klein - little
klein - tiny
klein - small
blau - blue
Meer - sea
Meer - ocean
ist - is
```

B.2 Interaktive Benutzerschnittstelle

Nach dem Start nimmt das Programm über die Konsole Befehle unter Verwendung der Methode `readLine` Klasse `Terminal` entgegen. Nach der Verarbeitung eines Befehls wartet das Programm auf weitere Eingaben, bis es durch Eingabe des `quit`-Befehls beendet wird. Die Eingabe und die Ausführung der Befehle dürfen nicht gegen die definierten Spezifikationen verstoßen. Bei einem Verstoß muss immer eine aussagekräftige Fehlermeldung ausgegeben, anschließend wartet das Programm regulär auf weitere Eingaben. Beachten Sie, dass nicht bei jeder erfolgreichen Interaktion automatische eine Ausgabe erfolgen muss.

B.2.1 Terminalsymbole

`<word> <translated_word>` Wörter aus der Start- bzw. der Zielsprache.

`<letter>` Ein Buchstabe aus dem deutschen Alphabet.

`<sentence>` Satz bestehend aus mehreren Wörtern die durch Leerzeichen voneinander getrennt sind. Es sind keine Satzzeichen erlaubt.

B.2.2 Befehle

`add <word> <translated_word>` Fügt das eingegebene Wortpaar dem Wörterbuch hinzu, wenn dieses Wortpaar noch nicht gespeichert wurde.

`remove <word>` Entfernt ein Wort und alle zugehörigen Übersetzungen aus dem Wörterbuch, wenn dieses Wort dort existiert.

`print` Im Erfolgsfall wird jedes Wort zeilenweise zusammen mit seinen Übersetzungen ausgegeben. Das Wort aus der Startsprache wird durch den Separator von einem oder mehreren Wörtern aus der Zielsprache getrennt. Dabei werden alle Wörter aus dem Wörterbuch zusammen mit ihren Übersetzungen in alphabetischer Reihenfolge aufgelistet. Wenn ein Wort mehr als eine Übersetzung hat, werden diese Wörter alphabetisch geordnet und durch ein Komma getrennt. Wenn keine Wörter ausgegeben werden können, wird nur eine Fehlermeldung ausgegeben.

`print <letter>` Die Ausgabe dieses Befehls entspricht weitgehend der des parameterlosen `print`-Befehls. Im Erfolgsfall wird jedes Wort, das mit dem Buchstaben (Groß-/Kleinschreibung wird nicht berücksichtigt) beginnt, Zeile für Zeile zusammen mit seinen Übersetzungen ausgegeben. Alle Wörter aus dem Wörterbuch werden zusammen mit ihren Übersetzungen, die mit dem angegebenen Buchstaben beginnen, aufgelistet. Wenn es im Wörterbuch keine Wörter gibt, die mit diesem Buchstaben beginnen, wird nur eine Fehlermeldung ausgegeben.

`translate <word>` Im Erfolgsfall gibt dieser Befehl alle Übersetzungen des eingegebenen Wortes aus. Dabei wird die Groß- und Kleinschreibung der Eingabe berücksichtigt. Wenn für das Wort mehrere Übersetzungen gespeichert wurden, werden diese Wörter alphabetisch geordnet und durch ein Komma getrennt in einer Zeile ausgegeben.

`translate <sentence>` Mit diesem Befehl wird der eingegebene Satz übersetzt, indem Wort für Wort die entsprechende Übersetzung aus dem gespeicherten Wörterbuch ausgegeben wird. Im Erfolgsfall werden alle möglichen Kombinationen von Übersetzungen des Satzes zeilenweise ausgegeben. Bei der Eingabe wird zwischen Groß- und Kleinschreibung unterschieden, und die Zeilen der Ausgabe werden alphabetisch geordnet. Wenn für mindestens eines der Wörter des eingegebenen Satzes keine Übersetzung im Wörterbuch gefunden wird, wird nur eine Fehlermeldung ausgegeben.

`quit` Der Befehl beendet das Programm vollständig.

B.3 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Die Eingabezeilen werden mit dem Größer-als-Zeichen gefolgt von einem Leerzeichen eingeleitet, diese beiden Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen nur der Unterscheidung zwischen Ein- und Ausgabezeilen.

Für die folgende Beispielinteraktion wurde das Kommandozeilenargument zum Pfad der in Abschnitt B.1 gezeigten Textdatei angegeben, d.h. in diesem Beispiel enthält das Wörterbuch bereits die dort elf angegebenen Wortpaare.

Beispielinteraktion

```
1  > print d
2  Das - The
3  Der - The
4  Die - The
5  > translate Der
6  The
7  > translate klein
8  little,small,tiny
9  > add Haus Home
10 > translate Haus
11 Home,House
12 > remove Haus
13 > print
14 blau - blue
15 Das - The
16 Der - The
17 Die - The
18 ist - is
19 klein - little,small,tiny
20 Meer - ocean,sea
21 > translate Das Meer ist blau
22 The ocean is blue
23 The sea is blue
24 > quit
```

Aufgabe C: Kopfgeldjägerin (7 Punkte)

Die Kopfgeldjägerin *Romana Antiheld* verfolgt einen flüchtigen Verbrecher quer durch das Universum. Glücklicherweise haben ihr Hyperraumsprünge es sehr erleichtert, mehrere Planeten P schnell zu besuchen. Hierbei verfügt jeder Planet $p_n \in P$ über eine Reihe von Portalen in das Hyperraumnetzwerk $R \subseteq P \times P$. Jedes Portal ist mit einem Portal auf einem anderen Planeten verbunden. Aus offensichtlichen Sicherheitsgründen sind diese Hyperraumrouten $(x, y) \in R$ nur in eine Richtung passierbar, wobei ein Portal der Eintrittspunkt x und das andere Portal der Austrittspunkt y aus dem Hyperraum ist. Zur Vereinfachung können Sie zusätzlich annehmen, dass für alle Planeten der Eingangsgrad $d_p^-(p)$ größer oder gleich dem Ausgangsgrad $d_p^+(p)$ ist $(\forall p : d_p^-(p) \geq d_p^+(p))$ und wenn ein Planet einen Eingangsgrad von 0 hat, hat er einen maximalen Ausgangsgrad von 1 $(\forall p : d_p^-(p) = 0 \Rightarrow d_p^+(p) \leq 1)$. Darüber hinaus muss das Netz der Hyperraumrouten zyklenfrei sein, um Unfälle bei Hyperraumsprüngen zu vermeiden. Das heißt, es darf im Hyperraumnetzwerk keinen Weg $p_1, \ldots, p_n$ für $i = 1, \ldots, n$ mit $p_1 = p_n$ geben.

Beim Betrachten ihrer Sternenkarte fragt sich *Romana Antiheld*, wie viele Droiden sie für eine Suche auf jedem Planeten brauchen würde. Die Droiden müssen von verschiedenen Planeten zur gleichen Zeit aus starten, sonst könnte der Verbrecher misstrauisch werden und fliehen. Zwar kann jeder Droid von einem Planeten seiner Wahl aus starten und sich auf den Hyperraumrouten von Planet zu Planet fortbewegen, doch ist er nach wie vor an die Grenzen des Hyperraumnetzes gebunden.

C.1 Eingabe

Ihr Programm nimmt als einziges Kommandozeilenargument einen Pfad auf eine Textdatei entgegen. Verwenden Sie die Methode `readFile` der Klasse `Terminal`, um den Inhalt dieser Datei zu lesen. Hierbei können Sie nicht davon ausgehen, dass die gegebene Eingabe nach der gegebenen Spezifikationen entspricht. Fehlerhafte Eingaben führen zum Beenden des Programms, nachdem eine passende Fehlermeldung ausgegeben wurde.

Die Textdatei beginnt mit einer ganzen Zahl N $(1 < N \leq 1000)$, welche die Anzahl der Planeten festlegt. Die Planeten P werden von 0 bis $N - 1$ durchnummeriert. Die folgenden N Zeilen geben die Hyperraumrouten an. Die i-te dieser Zeilen enthält zuerst die Anzahl der Verbindungen K $(0 \leq K \leq N - 1)$ vom Planeten i, gefolgt von K ganzen Zahlen, welche die Zielplaneten angeben.

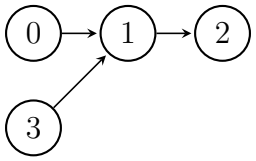
C.2 Ausgabe

Geben Sie die Mindestanzahl von Droiden aus, die für den Besuch jedes Planeten benötigt wird.

C.3 Beispielablauf

C.3.1 Beispieleingabe 1

Veranschaulichung der Beispieleingabe



Textdatei der Beispieleingabe

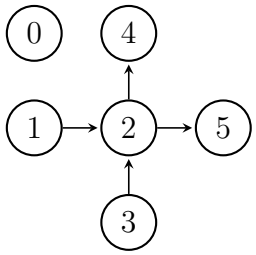
```
4
1 1
1 2
0
1 1
```

Ausgabe für gegebene Textdatei

```
2
```

C.3.2 Beispieleingabe 2

Veranschaulichung der Beispieleingabe



Textdatei der Beispieleingabe

```
6
0
1 2
2 4 5
1 2
0
0
```

Ausgabe für gegebene Textdatei

```
3
```

Aufgabe D: Typkonvertierung (2 Punkte)

Schreiben Sie die unten angegebene Java-Klasse `TypeConversion` zunächst ab. Schauen Sie sich dann das gegebene Programm genau an. Analysieren Sie mit Ihrem Wissen aus der Vorlesung, welche Typkonvertierungen vorkommen und von welcher Art die Typkonvertierungen sind. Annotieren Sie dann jeweils per Kommentar:

1. Welche der Konvertierungsarten jeweils in der Zeile vorkommen:
   a) *Widening Conversion*
   b) *Narrowing Conversion*
   c) *String Conversion*
   d) *Casting Conversion*
2. Welche expliziten Konversionen auch implizit (d.h. ohne explizites Casting) stattgefunden hätten.

```
1  package edu.kit.informatik;
2  public class TypeConversion {
3      public static void main(String[] args) {
4
5          int i = 42;
6
7          byte b = (byte) (0xFF & (i << 24));
8
9          int j = 'a' + 42;
10
11         System.out.println("Some letters are ... " + 'b' + 'c');
12
13         short s = 42;
14
15         char c = (char) s;
16
17         long l = 99999999999999L;
18
19         double d = 19.0f;
20
21         d += l;
22
23         long l2 = (long) (i * 2.0d) + (j + s);
24
25         float[] Array = { 0xFF, l2, (float) d };
26
27         String o = (String) Array.toString();
28
29         System.out.println(Array[0]);
30     }
31 }
```