Projekt P1: Vokabeltrainer [1]
(40 = 20 + 10 + 10 Punkte)

Themen: Collections, Streams, Exceptions, XML

In diesem Projekt sollen Sie einen Vokabeltrainer entwickeln, mit dem es möglich ist:

- neue Vokabeln (strukturiert) einzufügen und in einer Datei zu speichern,
- Vokabeln zu suchen und ggf. zu ändern und
- eine Vokabelabfrage durchzuführen.

Überblick:

Die Projekt-Struktur beinhaltet vier Pakete, die im Folgenden beschrieben werden:

Package `model` Dieses Package umfaßt die Klassen zur Repräsentation und Verwaltung der (Vokabel-)Daten. Hierzu gehören zunächst folgende Klassen:

- Die Klasse `GlobalData` enthält neben zwei Konstanten für die hier verwendeten Locales (Englisch und Deutsch) eine Beispiel-Vokabel und zwei weitere, statische Attribute, auf die von mehreren Komponenten aus zugegriffen wird: dies ist zum einen die Vokabelliste `currentLoadList` mit der Liste der jeweils aktuell verfügbaren Vokabeln (geladene und evtl. ergänzte Liste) sowie die Variable `searchVocable`, welche eine Vokabel referenziert, die bei einer Suche (mit dem Suchfenster) gefunden und dann ggf. genauer angesehen oder geändert werden soll (mit dem Fenster zur Eingabe von Vokabeln).
- Die Klasse `Vocable` definiert eine Datenstruktur für einen Vokabel-Eintrag, bestehend aus einem Wort in einer Quellsprache, einer Liste von möglichen Übersetzungen in einer Zielsprache sowie Beispielen, in denen dieses Wort verwendet wird. Darüberhinaus kann

[1] Dieses Projekt basiert auf einer Seminararbeit von Simon Ermler

eine Vokabel einer Unit und einer Section zugeordnet werden. Das Attribut `lFactor` dient dazu, bei der Vokabel-Abfrage den Wissensstand in Bezug auf diese Variable zu repräsentieren (Lernstand). Neben mehreren Konstruktoren sowie Getter- und Setter-Methoden für die Attribute ist eine `toString`-Methode für die kompakte Darstellung einer Variablen in einer `JList`-Instanz vorgegeben (wird in der `SearchGUI`-Instanz verwendet). Die Methode `getRandomTranslation` ermöglicht darüber hinaus die zufällige Auswahl von einer der für eine einzelne Vokabel gespeicherten Übersetzungen.

- Die Klasse `VocableList` kapselt eine Liste von Vokabeln, die in einer Sammlung (`Collection`) gehalten werden. Hier sind neben zwei Konstruktoren und einer Getter-Methode für die Liste mehrere Methoden vorgegeben, die einen indirekten Zugriff auf diese Liste zulassen:
  - Die Methode `add` ermöglicht das Hinzufügen einer `Vocable`-Instanz in die Liste
  - die Methode `size` liefert die aktuelle Anzahl der Einträge in der Liste
  - die Methode `contains` überprüft, ob eine bestimmte `Vocable`-Instanz in der Liste enthalten ist (Referenz-Vergleich).
- Das Interface `LearningContext` spezifiziert Methoden für den Zugriff auf ein Objekt, welches die Rahmenbedingungen für eine Vokabelabfrage beinhaltet (siehe Aufgabenstellung).
- Das Interface `AnswerCollection` spezifiziert Methoden für den Zugriff auf ein Objekt, welches die Antworten der Abfrage durch die `LearnGUI`-Instanz sammelt und auch die Auswertung übernimmt (siehe Aufgabenstellung).

Package `io` Dieses Package umfaßt alle Klassen, die mit einer Ein- und Ausgabe der Vokabellisten Import/Export) zu tun haben. Vorgegeben sind hierbei die folgenden Klassen:

- Die Klasse `IOSettings` beinhaltet zunächst zwei statischen Konstanten, die für die Trennung von Einträgen in einer Zeile bei einem tabellenorientierten Dateiformat (z.B. CSV) benötigt werden. Die Methode `getLocale` liefert eine `Locale`-Instanz, die zu

einer als Parameter übergebenen Codierung für die Sprache paßt. Diese wird nowendig, da in den Ausgabedateien stets nur die Sprach-Codierungen (z.B. "eng" für englisch) gespeichert werden, die beim Einlesen immer auf Locale-Instanzen abgebildet werden müssen.

- Die abstrakte Klasse `LineReader` bietet einen Rahmen zum Einlesen einer Datei, in der die Daten zeilenweise organisiert sind wie z.B. in einem CSV Format. Die Methode `readFile` öffnet eine Datei mit dem als Parameter übergebenen Dateinamen und liest den Inhalt der Datei zeilenweise ein. Die abstrakte Methode `parseLine` ist dabei dafür zuständig, eine einzelne Zeile zu verarbeiten und (in diesem Fall) eine Instanz der Klasse `Vocable` zu generieren und zurückzugeben. Je nach verwendetem Format muss die Methode in einer konkreten Unterklasse passend implementiert werden.
- Die abstrakte Klasse `VocableWriter` dient als Rahmen für einen Export einer Vokabelliste in eine Datei. Die Methode `writeFile` öffnet eine Datei, deren Name als Parameter übergeben wird, zum Schreiben. Das Schreiben der einzelnen Vokabeln aus der ebenfalls als Parameter übergebenen Liste muss, je nach Format, in einer konkreten Unterklasse durch die Methoden `mapVoc` und `writeAll` implementiert werden.
- Die abstrakte Klasse `XMLWriter` bietet einen Rahmen zum Exportieren einer Vokabelliste in eine `XML`-Datei. Die in einer konkreten Unterklasse zu implementierenden Methoden sind `getVocableElements` (welche eine Sammlung aller XML-Elemente generieren und zurückgeben soll, die eine einzelne Vokabel repräsentieren) und `parseVocableElement` (die aus einem XML-Vokabel-Element eine Instanz der Klasse `model.Vocable` erzeugen und zurückgeben soll).

Package `view` Dieses Package enthält alle Klassen, aus deren Instanzen die Graphische Oberfläche erstellt wird. Dazu gehören auch die Listener, die meist in anonymer Form an die Interaktionselemente (Buttons, Comboboxen, Menuitems etc.) gebunden sind. Dabei werden alle Operationen, die sich nicht nur lokal abspielen (wie z.B. das Löschen des Texts in einem Eingabefeld), über Kommandocodes an die zugehöri-

gen Controller-Instanzen weitergeleitet. Jeder GUI-Klasse ist dabei ein Controller zugeordnet. Das Startfenster, von dem sich alle weiteren Aktionen ergeben, gehört der Klasse `OverviewGUI` an und ist einer Instanz der Klasse `MainController` zugeordnet. Das Fenster, mit dem neue Vokabeln eingefügt bzw. geändert werden können (sollen), ist eine Instanz der Klasse `InsertGUI` und kommuniziert mit einer Instanz der Klasse `InsertController`. Die Klasse `SeachGUI`, mit deren Hilfe Vokabeln gesucht werden sollen, arbeitet mit einem Controller der Klasse `SearchController` und das Fenster, welches die Möglichkeit bereitstellt, interaktiv Vokabeln abzufragen, arbeitet mit einem Controller der Klasse `LearnController`. Fenster- und Controller-Instanz besitzen sich dabei jeweils gegenseitig als Attribut. Die Klasse `GUISettings` definiert schließlich ein paar Farben und einen Font für die verschiedenen Fenster.

Package `Controller` Die Klassen in diesem Package sind für den Datenfluß zwischen den einzelnen Klassen und der GUI zuständig. Hierzu ist ein Interface `Controller` vorgegeben, welches neben Konstanten für alle vorkommenden Kommandocodes (z.B. `ADD`, `EDIT` oder `EXIT`) drei Methoden definiert:

- `void startController()`
- `void stopController()`
- `void execCommand(String cmd)`

Alle vorgegebenen Controller-Klassen implementieren dieses Interface.

Die ersten beiden Methoden sorgen hauptsächlich für die Generierung der Controller- und der zugehörigen GUI-Instanzen (`startController`) bzw. deren Freigabe beim Beenden des Programms (`stopController`) sowie dafür, dass die Fenster sichtbar bzw. unsichtbar gemacht werden.

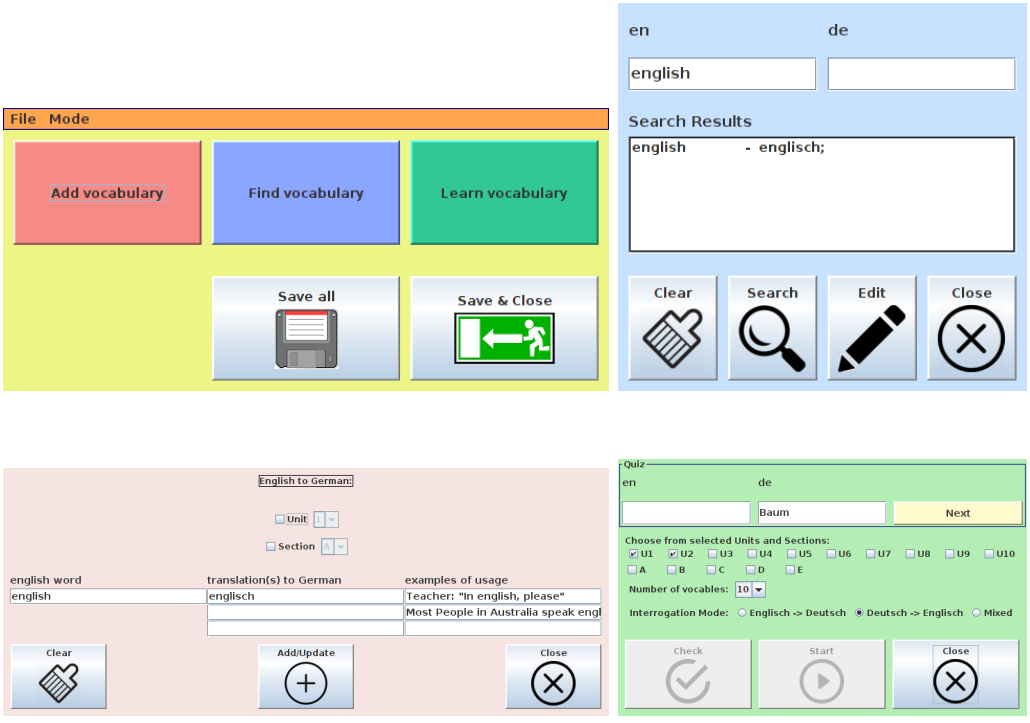
Die Methode `execCommand` führt mit Hilfe des als Parameter übergebenen Kommandocodes die dem jeweiligen Controller zugeordneten Operationen aus.

Wie oben beschrieben gibt es zu jeder GUI-Klasse einen passenden Controller.

Ergänzend hierzu stellt die Klasse `Settings` ein paar (vorwiegend statische und konstante) Attribute zur Verfügung, die

- einen Modus für Datei-Operationen festlegen (CSV oder XML),
- (Default-)Sprachen für die Vokabelliste definieren,
- maximale Anzahlen für die Units und Sections festlegen,
- Pfadnamen für die Bilder beinhalten, die in den GUIs verwendet werden sowie für den Default-Folder zum Speichern von exportierten Daten definieren,
- mehrere Methoden, um Dateinamen zum Importieren und Exportieren festzulegen bzw. diese zu holen
- eine Filter-Klasse für die Klasse `JFileChooser` definiert, mit der sich verschiedene Dateitypen (CSV/XML) ausfiltern lassen.

Die folgenden Bilder zeigen Screenshots der verschiedenen Fenster (oben die Übersicht und das Suchfenster, unten das Fenster zum Einfügen und die Lernkomponente):



Alle Bilder und andere Daten (wie z.B. csv- und XML-Dateien) werden im Ordner `resources` gespeichert. Sollte Sie weitere Bilder hinzufügen wollen,

sind diese ebenfalls hier abzulegen. Darüberhinaus liegt in diesem Verzeichnis eine CSV-Datei mit Beispiel-Vokabel-Einträgen zum Testen.

Aufgabenstellung:

(a) Im ersten Schritt geht es darum, die Liste der Vokabeln (`GlobalData.currentLoadList`) zu verwenden, um darin neue Vokabeln einzutragen und diese zu filtern (nach bestimmten Suchmustern). Gehen Sie dazu wie folgt vor:

- Entwickeln Sie (im Package `model`) eine Klasse, die das Interface `LearningContext` implementiert. Diese Klasse soll Attribute besitzen, die einen Lernkontext repräsentieren, wie er durch die GUi hierzu spezifiziert werden kann. Hierzu gehören die Units und Sections, aus denen die Vokabeln gewählt werden, die (maximale) Anzahl der Vokabeln in einer Vokabelliste für eine Abfrage sowie der Abfrage-Modus (*englisch* → *deutsch*, *deutsch* → *englisch* oder gemischt). Legen Sie passende Attribute und einen Konstruktor zur Initialisierung der Attribute an. Passen Sie dann den Konstruktoraufruf in der Klasse `LearnGUI` (Zeile 254) entsprechend an und entfernen Sie die Kommentarzeichen davor. Implementieren Sie dann die Methoden des Interfaces:
  - `public int getMode()`
    Die Methode soll den Abfragemodus für die nächste Vokabel zurückgeben, d.h. einen der Werte `SOURCE_GIVEN` oder `TARGET_GIVEN`, die in der Klasse `LearnGUI` spezifiziert sind. Falls der in der Instanz der Klasse gesetzte Modus den Wert MIXED hat, soll ein Zufallswert (`SOURCE_GIVEN` oder `TARGET_GIVEN`) zurückgegeben werden.
  - `public int getSize()`
    Diese Methode soll die gewünschte Größe des Lerndatensatzes (die Anzahl der Vokabeln, die abzufragen sind) zurückgeben.
  - `boolean matches(Vocable v)`
    Diese Methode soll prüfen, ob eine einzelne Vokabel die Bedingungen erfüllt, die durch den aktuellen Lernkontext definiert werden und dementsprechend den Wert `true` bzw. `false` zurückgeben. Hierzu gehört beispielsweise, ob die Vokabel in einer der ausgewählten Units liegt.

- Entwickeln Sie (im Package `model`) eine Klasse, die das Interface `AnswerCollection` implementiert. Diese Klasse soll die gegebenen Antworten (die sich aus je einem String für jede abgefragte Vokabel und dem jeweiligen lernmodus rekrutieren und durch die Methode `addAntwort` übermittelt werden), sammeln. Um einen gemischten Lernmodus zu realisieren, muss für jede Antwort auch der jeweilige konkrete Modus gespeichert werden. Im gemischten Modus wird dann der Klasse durch die Methode `addAnswer` auch der in diesem Fall angewendete Modus (`SOURCE_GIVEN` oder `TARGET_GIVEN`) übermittelt, welcher bei der Auswertung später dazu dient, herauszufinden, ob sich die Antwort auf den Begriff in der Quell- oder in der Zielsprache bezieht. Implementieren Sie dann die Methoden des Interfaces:
  - Die Methode `addAnswer` trägt eine Antwort und den zugehörigen Modus in die Liste der Antworten ein.
  - Die Methode `check` Diese Methode wird normalerweise nur aufgerufen, nachdem alle Antworten eingegeben und hinzugefügt worden sind. In der Methode sollen dann die gegebenen Antworten mit den Vorgaben aus der als Parameter übergebenen Vokabelliste verglichen und ein Ergebnis angezeigt werden. Sie können davon ausgehen, dass die i-te Antwort (die beim i-ten Aufruf von `addAnswer` eingetragen wurde) der i-ten Vokabel in dem Vektor entspricht. Überlegen Sie sich dazu wie ein passender Vergleichswert berechnet werden kann. Unterscheiden Sie dabei auch zwischen leichten Unterschieden zwischen Vorgabe und Antwort (z.B. nur Groß/Kleinschreibung oder Antwort ist Teil der Vorgabe) und komplett falschen bzw. fehlenden Antworten (durch ein leeres Wort). Verwenden Sie den Vergleichswert, um den Lernfaktor entsprechend anzupassen (bei richtiger Antwort soll der Wert erhöht, bei falscher Antwort verringert werden). Sammeln Sie alle Antworten zusammen mit den Lösungen und den (Teil-)Bewertungen und zeigen Sie diese inklusive eines Ergebnisse (z.B. eine Note oder eine erreichte Punktzahl im Vergleich zu einer maximalen Punktzahl) am Ende grafisch (z.B. mit Hilfe einer Unterklasse von `JDialog`, in der ein entsprechend formatierter String angezeigt wird) an. Passen Sie auch den

"Lernfaktor" der abgefragten Variablen an: verringern Sie den Wert bei Fehlern und erhöhen Sie ihn bei korrekten Antworten.

- Die Methode `getCurrentIndex` soll den Index der nächsten erwarteten Antwort zurückgeben. Diese Methode wird hier nur von der Klasse `LearnController` verwendet um die nächste Vokabel aus der lokalen Liste mit den abzufragenden Vokabeln zu holen.

Passen Sie den Konstruktor-Aufruf in der Klasse `LearnController` (Zeile 42) entsprechend an und entfernen Sie die Kommentarzeichen davor.

- Erweitern Sie die beiden unvollständigen Methoden in der Klasse `model.VocableList` wie im Folgenden beschrieben:
  - `public Collection<Vocable> chooseLearningList (LearningContext ls)`
    Diese Methode soll eine Sammlung von Vokabeln erstellen und in einem Vektor zurückgeben, die den Bedingungen entspricht, die in dem als Parameter übergebenen Lernkontext spezifiziert ist. Falls möglich (falls genug vorhanden), soll die Anzahl der Vokabeln in der Sammlung der in dem Lernkontext spezifizierten Anzahl entsprechen. Sorgen Sie dafür (z.B. durch Mischen der Ergebnisliste), dass die Vokabeln nicht in der Reihenfolge zurückgegeben werden, die sie in der Liste `GlobalData.currentLoadList` haben. Bevorzugen Sie bei der Auswahl Vokabeln mit einem niedrigen Wert für den Lernfaktor (so dass Vokabeln, die noch nicht beherrscht werden, häufiger abgefragt werden).
  - `public Collection<Vocable> find(String w1, String w2)`
    Diese Methode wird für die Suche nach Vokabeln benötigt. Die Parameter `w1` und `w2` spezifizieren zwei (Teil-)Worte in der Quell- bzw. Zielsprache, nach denen gesucht werden soll. Der Wert null bzw. ein leerer String soll dabei bedeuten, dass kein Suchbegriff spezifiziert wurde. Die Methode soll dann alle Vokabeln ausfiltern und in einer Liste zurückgeben, bei denen das (nichtleere) Wort `w1` ein Teilstring des Worts in der Quellsprache ist oder bei denen das Wort `w2` ein Teil (mindestens) einer der definierten Übersetzungen ist.

(b) Erweitern Sie das Package `io` um zwei Klassen zum Lesen bzw. Schreiben von Vokabellisten aus einer CSV-Datei bzw. in eine CSV-Datei. Leiten Sie die Klassen dabei von den abstrakten Klassen `LineReader` bzw. `VocableWriter` ab. Sammeln Sie die String-Einträge beim Rausschreiben dabei in einer `StringBuffer`-Instanz. Eine Zeile in einer CSV-Datei mit einer Vokabelliste (mit Spalten für die Quellsprache, die Zielsprache, einen Begriff in der Quellsprache, ein oder mehrere, durch ein Trennzeichen voneinander separierte Übersetzungen des Begriffs in die Zielsprache, ein oder mehrere, durch ein Trennzeichen voneinander separierte Beispiele, zwei (optionalen) Einträgen für die Unit und die Section sowie einer Zahl für das Lernstandsbewertung), könnte beispielsweise wie folgt aussehen:

```
eng;deu;tree;Baum;big plant with green leafs|Christmas tree|CS:
a data structure for  hierarchies;1; ;0.0
```

Überlegen Sie sich nun noch drei verschiedene Fehlertypen, die, z.B. bei einer manuell erstellten CSV-Datei auftreten können. Schreiben Sie dazu entsprechende Exception-Klassen (mit einer gemeinsamen Oberklasse) und fangen Sie die Fehler ab. Zeilen mit einem Fehler sollen jeweils, zusammen mit einer passenden Meldung auf der Konsole ausgegeben und dann im Weiteren ignoriert werden. Sie dürfen hier auch die abstrakte Klasse `LineReader` abändern, um Ausnahmen in der Methode `readFile` verarbeiten zu können.

(c) Erweitern Sie das Package `io` um zwei Klassen zum Lesen bzw. Schreiben von Vokabellisten in einem XML-Format. Leiten Sie die Klassen dabei von den abstrakten Klassen `XMLVocableReader` und `XMLVocableWriter` ab. Überlegen Sie sich hierzu zunächst ein passendes Format und schreiben Sie eine DTD (die Sie im Verzeichnis `resources` ablegen können). Implementieren Sie dann die fehlenden Methoden, so dass das von Ihrer DTD verwendete Format gelesen und geschrieben werden kann. Passen Sie die Aufrufe in den Methoden `loadFile` und `saveToFile` entsprechend der gewählten Klassennamen an und entfernen Sie zum Testen die Kommentarzeichen. Erweitern Sie die Liste der Importe in der Klasse entsprechend.

Anmerkungen und Hinweise

Bei Bedarf dürfen Sie die zu ergänzenden Klassen mit weiteren privaten Attributen und Methoden ausstatten.