Projekt 10 Der Vokabeltrainer – mit der List unterwegs

Ein Vokabelttrainer soll programmiert werden. 16 Vokabeln sind vorgegeben. Von einer Vokabel wird jeweils das deutsche Wort vorgegeben und nach der Übersetzung gefragt. Ist die Übersetzung richtig, wird die Vokabel gelöscht und nach der nächsten Vokabel gefragt. Ist die Übersetzung falsch, wird die Vokabel ans Ende angehängt und später nochmals nach ihr gefragt. Man kann aber auch die Reihenfolge der Vokabelabfrage verändern: man kann eine Vokabel überspringen oder zur vorherigen zurückkehren. Außerdem kann man zusätzliche Vokabeln in die Vokabelliste eingeben.

Als Datenstruktur eignen sich hier weder die Schlange (**Queue**) noch der Stapel (**Stack**), da man hierbei jeweils nur auf das erste Element zugreifen kann. Viel besser geeignet ist hier die Datenstruktur der Klasse Liste (**List**), in der die einzelnen Knoten wieder nacheinander angelegt werden. Man kann auf das erste Element (**first**) und das letzte Element (**last**) zugreifen. Zusätzlich gibt es ein aktuelles Element (**current**), das man stückweise von vorne nach hinten bewegen kann.

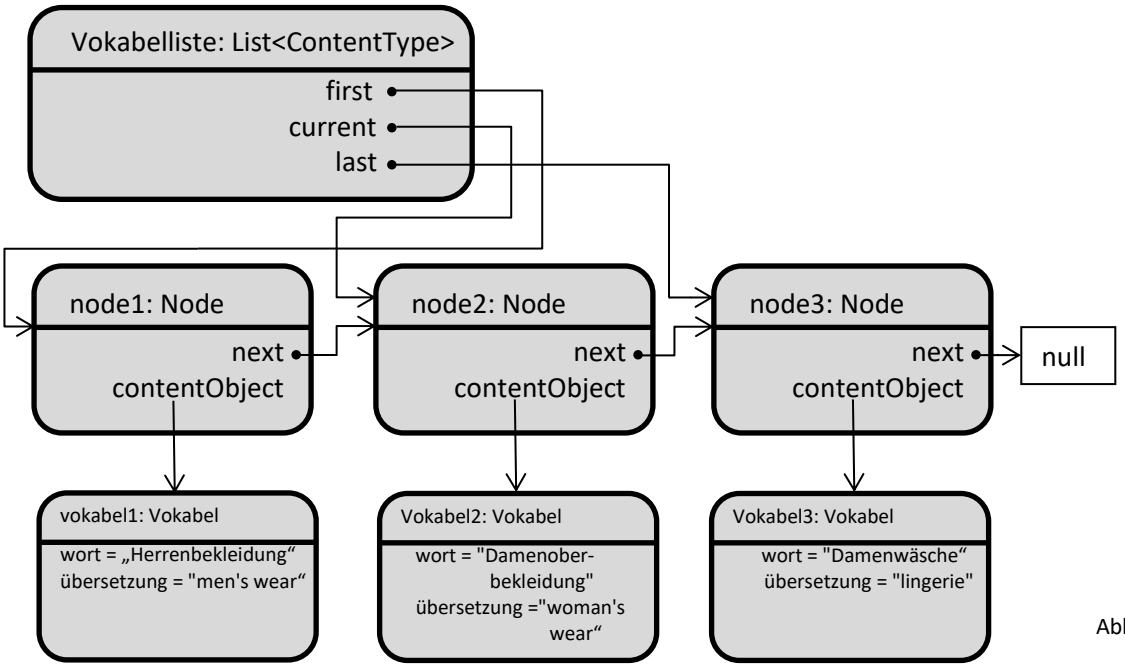

Abbildung 1

Aufgabe 1: Kopiere das Projekt Vokabeltrainer aus dem Schülerordner in deinen persönlichen Ordner und öffne das Projekt in Netbeans. Du siehst in der Klasse Trainer eine fertige Oberfläche, mit der man die Funktionen des Vokabeltrainers aufrufen kann. Implementiere als erstes die Klasse Vokabel, in der jeweils ein deutsches Wort (**wort**) mit seiner Übersetzung (**übersetzung**) gespeichert wird. Wie üblich benötigen wir den Konstruktor zum Anlegen einer Vokabel und zwei get-Methoden, wie im Klassendiagramm angegeben.

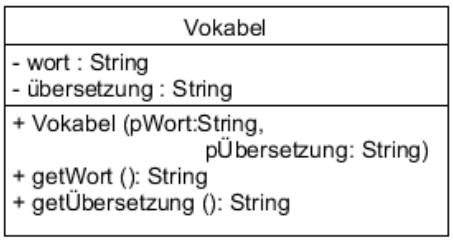

Abbildung 3

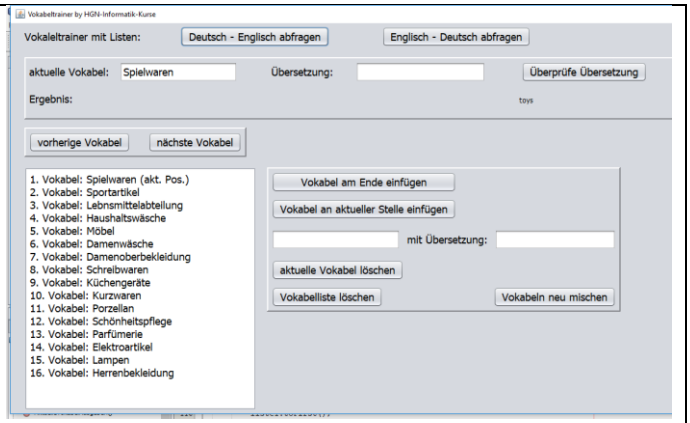

Abbildung 2

Wenn du das Projekt nach Fertigstellung der Klasse Vokabel startest, sollte die gesamte Vokabelliste wie angegeben erscheinen. Wenn du in den Konstruktor der Klasse Trainer schaust, siehst du, wie zunächst eine die list1 vom Typ List<Vokabel> erzeugt wird. Anschließend wird aus zwei String-Arrays die Liste aufgebaut. Dabei wird eine erste Methode liste1.append(vokabel1) benutzt, die einen Knoten am Ende der Liste mit dem Inhalt einer Vokabel einfügt. Welche Methoden die Klasse List außerdem zur Verfügung stellt siehst du in der folgenden Abbildung 4:

Aufgabe 2: Wir beginnen zunächst mit der Übersetzung Deutsch-Englisch, d. h. im **JTextField aktuelleVokabelBox** wird immer eine deutsche Vokabel angezeigt, im **JTextField aktuelleÜbersetzungBox** soll der Benutzer die Übersetzung angeben. Implementiere zunächst die Methode **überprüfeÜbersetzung-ButMouseClicked**: Klickt der Benutzer auf den entsprechenden Button, soll der Vokabeltrainer prüfen, ob die Vokabel richtig übersetzt wurde. Ist das Ergebnis korrekt soll etwa die Meldung „Richtig: Die Übersetzung von Lampen ist ligthing." Im Label **ergebnisLab** ausgegeben werden. Die Vokabel wird aus der Liste gelöscht und die nächste Vokabel wird angezeigt. Ist das Ergebnis falsch, wird ein ähnliches Ergebnis angezeigt, die Vokabel wird gelöscht und am Ende der Liste eingefügt.
Die Methoden zum Löschen und Einfügen am Ende findest du rechts in Abbildung 4.

Benutzen kannst du in der Klasse Trainer die Methode **listeSchreiben (liste1)**, die die aktuelle Liste in der **jTextArea listenfenster** ausgibt. Außerdem gibt die Methode **aktuelleVokabelAusgeben()** das deutsche Wort der aktuellen Vokabel im **jTextField aktuelleVokabelBox** aus.

Tipp für den Vergleich von 2 Strings:
Für den Vergleich von 2 Strings kannst du die Methode string1.equals (string2) benutzen. Das Ergebnis ist eine boolscher Variablenwert der true ist, wenn die beiden Strings gleich sind.

Aufgabe 3: Die aktuelle Vokabel soll in der Liste nach vorne und hinten bewegt werden können. Vervollständige dazu die beiden vorhandenen Event-Methoden **vorherigeVokabelButMouseClicked** und **nächsteVokabelBut-MouseClicked**. Benutze die entsprechenden Methoden der Klasse Liste und gebe zum Schluss aktuelle Vokabel und Liste mit den bekannten Methoden der Klasse Trainer aus.
Tipp, um zur vorherigen Stelle zu gelangen:
Mit den folgenden Zeilen findet man heraus, an der wievielten Stelle das aktuelle Element steht:

```
Vokabel vokabel1 = liste1.getContent();
int i = 0;
liste1.toFirst();
while ( liste1.getContent() != vokabel1){
    i++;
    liste1.next();
}
```

Methoden der Klasse `List<ContentType>`

Konstruktor **`List()`** Eine leere Liste wird erzeugt.

Anfrage **`boolean isEmpty()`** Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

Anfrage **`boolean hasAccess()`** Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt,sonst liefert sie den Wert `false`.

Auftrag **`void next()`** Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d.h.`hasAccess()` liefert den Wert `false`.

Auftrag **`void toFirst()`** Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag **`void toLast()`** Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Anfrage **`ContentType getContent()`** Falls es ein aktuelles Objekt gibt (`hasAccess()==true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess()==false`) gibt die Anfrage den Wert `null` zurück.

Auftrag **`void setContent(ContentType pContent)`** Falls es ein aktuelles Objekt gibt (`hasAccess()==true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

Auftrag **`void append(ContentType pContent)`** Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess()==false`).Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

Auftrag **`void insert(ContentType pContent)`** Falls es ein aktuelles Objekt gibt (`hasAccess()==true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess()==false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess()== false`) und die Liste nicht leer ist oder `pContent==null` ist, bleibt die Liste unverändert.

Auftrag **`void remove()`** Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.

Auftrag **`void concat(List<ContentType> pList)`** Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList==null` oder eine leere Liste ist, bleibt die Liste unverändert.

Abbildung 4

Aufgabe 4: In den beiden **jTextFields namenBox** und **ÜbersetzungBox** kann der Benutzer eine neue Vokabel eingeben. Vervollständige die Event-Methoden **aktuellEinfügenButMouseClicked** und **hintenEinfügenButMouseClicked**, mit denen man die neue Vokabel entweder an aktueller Position oder am Ende der Liste einfügen kann. Schließlich fehlt noch die Implementation der Event-Methoden **vokabelLöschenButMouseClicked** und **listeLöschenButMouseClicked**, mit denen man entweder eine einzelne Vokabel an aktueller Position oder sogar die gesamte Vokabelliste löschen können soll.
Tipp zum Löschen der gesamten Liste: **while (liste1.hasAccess()) …**

Zusatzaufgabe 1: Im oberen Bereich der Gui findest du zwei Buttons, mit denen man die Richtung der Vokabelabfrage ändern kann. Implementiere die passenden Event-Methoden zum Umstellen der Richtung und setze die bereits deklarierte boolsche Variable **deutschEnglisch**. In den Methoden zur Listenausgabe muss jetzt in Abhängigkeit dieser Variablen die Textausgabe erfolgen.

Zusatzaufgabe 2: Erkläre an einem Beispiel, wie die Methode **mischen (List<Vokabel> liste1)** funktioniert.

Zusatzaufgabe 3: Es soll gezählt werden, wie viele Vokabeln bisher gewusst bzw. falsch beantwortet wurden. Ergänze dazu die entsprechenden Methoden.

.