Skript zur Unterrichtsreihe

Entwicklung eines Vokabeltrainers

Inhaltsverzeichnis:

Projekt „Minidatenbank“

Sicherlich kennst du die kleinen elektronischen Übersetzungscomputer, die einem zu den gängigsten deutschen Wörtern die passende englische Übersetzung liefern. Diese „Minidatenbanken“ haben ein Display, welches dir jeweils einen deutschen Begriff und dessen zugehörige Übersetzung anzeigen kann – mehr nicht. Dennoch sind im Innern des Computers mehrere tausend verschiedene Vokabeln gespeichert, welche du mit Hilfe von Tasten durchblättern kannst.

Im folgenden wollen wir eine solche Minidatenbank entwickeln, welches ungefähr so aussehen könnte:

Aufgabe 1: Entwerfe das Layout des Projekts. Welche Menüeinträge sollten unter dem Menüpunkt *Datei*, welche unter dem Menüpunkt *Datensatz* aufgeführt werden?

Einführung in die Projektidee

Bei der Entwicklung dieses Programms werden wir uns mit folgenden Fragen beschäftigen müssen:

- Welche Bedienfunktionen sollte unser Programm haben?
- Wie ist eine interne Speicherung der Daten möglich?
- Wie können die Daten auf längere Zeit gesichert werden (Datei speichern und öffnen)?
- Welcher Zusammenhang besteht zwischen den Daten im Hintergrund und dem Formular?

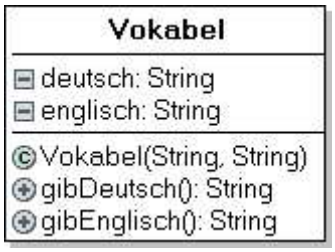
Fangen wir vielleicht am einfachsten mit der ersten Frage an. Welche Bedienfunktionen sind sinnvoll? Das Menü Datei steht im Normalfall für sämtliche Dateioperationen, also das Speichern und Laden einer ganzen Datei sowie der Einrichtung einer neuen und schließen einer vorhandenen Datei. Das Menü Datensatz ist für die einzelnen Einträge in der Datenbank zuständig. So sollte man neue Datensätze anlegen und bestehende Datensätze löschen können. Eventuell sollte man auch einen bereits vorhandenen Datensatz ändern oder nach einem vorhandenen Datensatz suchen können. Andere Funktionen wären noch denkbar, allerdings wollen wir uns auf diese vier beschränken. Damit wäre die erste Frage geklärt.

Die zweite Frage ist aus Informatikersicht zunächst interessanter. Du hast beim Lottoprojekt bereits eine Möglichkeit erfahren, mehrere Daten in einer Variablen zusammenzufassen – das Array in Java. Auch hier haben wir es mit mehreren Daten ein und des selben Typs zu tun,

welche in einer Variablen zusammengefasst werden sollen. Eine mögliche Variablendeklaration wäre also:

```
String[] deutsch = new String[100];
String[] englisch = new String[100];
```

Schön ist diese Art der Variablendeklaration allerdings nicht. Eigentlich gehören doch Name und Telefon zusammen und sollten als eine Einheit aufgefasst werden. Wir könnten nun natürlich das Array erweitern [1..200] und in den ersten einhundert Plätzen nur die Namen und in den Plätzen 101 bis 200 nur die Telefonnummern speichern. Eleganter ist es allerdings, einen Datensatz, bestehend aus der deutschen und englischen Übersetzung in einer Klasse zusammenzufassen. Diese Klasse nennen wir `Vokabel`, deren Objekte genau eine Vokabel aufnehmen können:



Aufgabe 2: Erstelle die Klasse Vokabel, die so zu dem abgebildeten UML-Diagramm passt.

Objekte dieser Klasse können aber wie gesagt nur eine einzige Vokabel enthalten. Wir wollten allerdings nicht nur einen Eintrag speichern, sondern mehrere. Deshalb kommt jetzt doch wieder unser Array ins Spiel, nun allerdings über den Datentyp `Vokabel`:

```
private final int MAX = 10; // feste Konstante
private Vokabel[] vokabeln = new Vokabel[MAX];
```

Aber was soll diese Konstantendeklaration in der ersten Zeile? Stelle dir vor, du stellst irgendwann fest, dass du mit 100 Speicherplätzen für deine gespeicherten Vokabeln nicht mehr auskommst. Dann müsstest du überall die Zahl 100 durch eine größere Zahl ersetzen. Hast du vorher eine Konstante definiert, so brauchst du nur dort einmal die Zahl zu erhöhen, und dein ganzes Programm läuft mit mehr als 100 Speicherplätzen.

Gut, Jetzt haben wir die Datenstruktur für die interne Speicherung klar:

`vokabeln:`

Affe monkey	Bär bear	…	...	…	Dose tin	…	...	Zucker sugar			

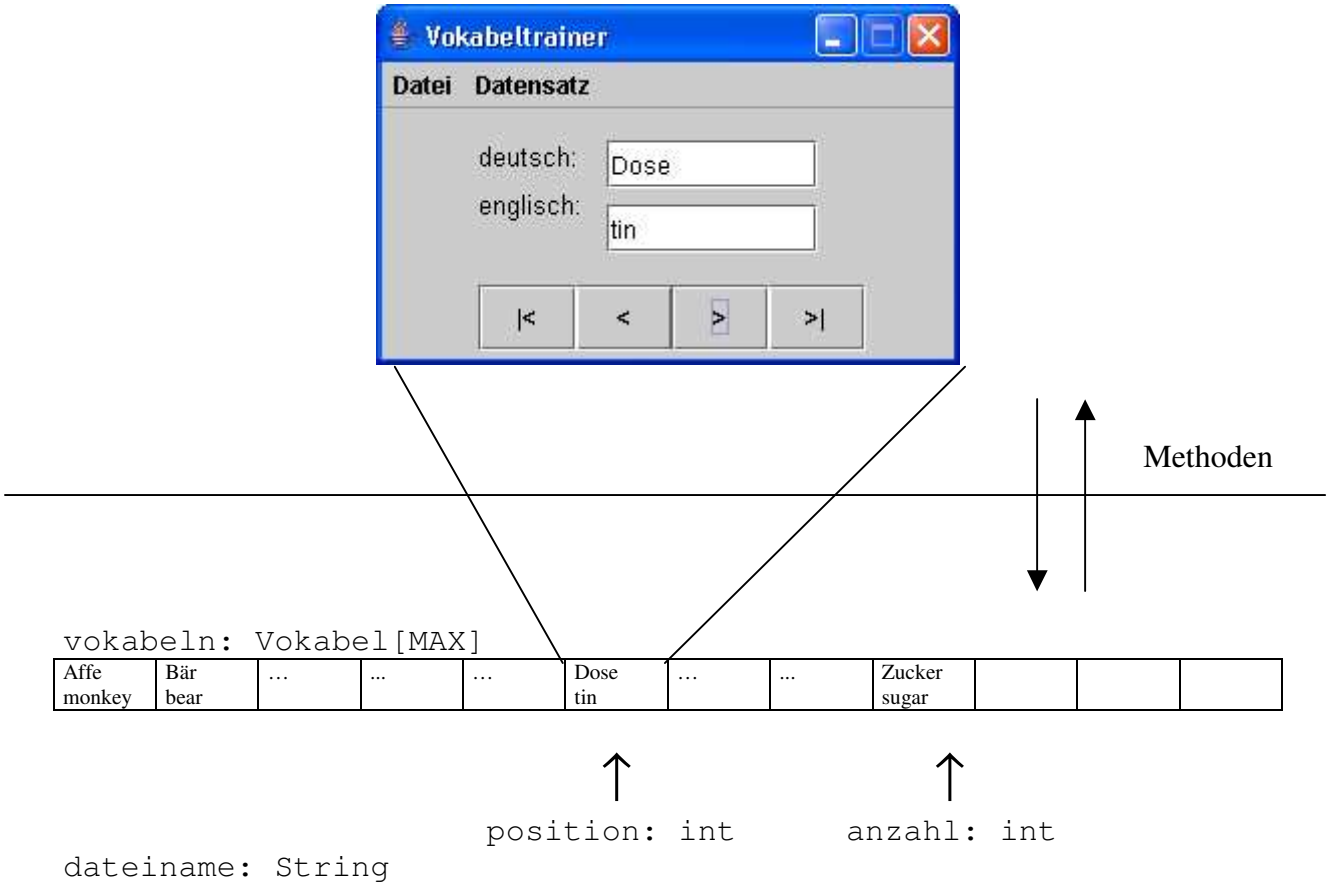
Wie du erkennen kannst, wird auf dem Formular immer nur ein aktueller Datensatz angezeigt. die anderen Datensätze verbleiben im Hintergrund. Woher weiß man allerdings, welches der aktuelle Datensatz ist? Irgendwie muss man sich doch seine **`Position`** merken, damit man im Array darauf zugreifen kann. Außerdem wäre es auch nicht schlecht zu wissen, welche **`Anzahl`** von Datensätzen insgesamt schon eingetragen wurde (die Konstante `MAX` gibt ja nur an, wie viele Datensätze höchstens gespeichert werden können). Eine weitere sinnvolle Informationen wären z. B. auch noch der **`Dateiname`**. Es ist üblich, den Dateinamen mitzuverwalten.

Insgesamt hätten wir also neben der Variablen `vokabeln` noch drei weitere Variablen (`position`, `anzahl` und `dateiname`). Diese vier Variablen gehören irgendwie zusammen, genauso wie die Daten einer Person (Name und Telefonnummer) zusammengehörten.

Aufgabe 3: Betrachte nochmals die vier Variablen, die zur Datenbank gehören. Welchem Datentyp gehören diese an?

Die Idee der Modularisierung

Das Rad muss nicht immer wieder neu erfunden werden – dieses Sprichwort hat auch in der Programmierung seine Bedeutung. Einmal geschriebene Teilprogramme sollten so aufgebaut sein, dass sie in anderen Programmen wiederverwendet werden können. Eine solche Datenbank, die im Hintergrund des Formulars existiert, könnte auch für andere Zwecke (z. B. Drucken eines Vokabelheftes, etc.) sinnvoll sein. Sie sollte also wiederverwendbar sein. Eine Operation wie das Suchen eines Begriffs in der Datenbank soll ja nicht nur von unserem Projekt aus funktionieren, sondern auch dann, wenn man unsere Datenbank in anderen Projekten nutzen will. Die „Operation“ oder „Methode“ Suchen gehört also logisch zur Datenbank und nicht zum Formular. Genauso ist es mit der „Methode“ Datei speichern. Der Benutzer klickt zwar im Menü auf Datei|Speichern..., allerdings ist es Sache der Datenbank, sich selbst zu speichern. Man müsste also der Datenbank den Befehl geben können: „Speichere dich unter dem Dateinamen xxx“. Ein anderer Befehl, den man der Datenbank geben können sollte, wäre z. B. auch: „Füge den deutschen Begriff xxx mit der englischen Übersetzung yyy in dir selbst ein“, oder: „Wie lautet dein aktueller Datensatz, auf dem du gerade stehst?“. Unsere Datenbank hat also nicht nur Variablen (die vier Stück von vorhin) sondern auch „Methoden“, welche die Kommunikation zwischen Formular und Datenbank herstellen. Bildlich kann man sich das wie folgt darstellen:



Aufgabe 4: Welche Operationen werden für die Kommunikation benötigt? Überlege dir bei jedem „Befehl“ an die Datenbank, welche Informationen du der Datenbank mitliefern musst bzw. welche Informationen du von der Datenbank bei einem „Befehl“ zurückerhalten möchtest.

Folgende Methoden wären nötig (sie entsprechen im wesentlichen den in Kapitel 1 erwähnten Menüpunkten):

```
public void speichern(File datei)
public void laden(File datei)
public void hinzufuegen(Vokabel neu)
public void loeschen()
public void aendern(Vokabel neu)
public boolean suchen(String deutsch)
```

Letztere Methode soll `true` oder `false` zurückgeben, je nachdem, ob der gesuchte deutsche Begriff in der Datenbank vorhanden war oder nicht.

Dies waren alles Methoden, welche die Kommunikation vom Formular zur Datenbank herstellen. Allerdings reicht das nicht aus, denn das Formular soll ja von der Datenbank auch Informationen zurückerhalten. Wir brauchen also auch noch die sogenannten getter-Methoden, um einerseits den aktuellen Datensatz an das Formular zurückgeben zu können und andererseits Informationen über Dateiname, Anzahl und Position zurückgeben zu können:

```
public Vokabel gibVokabel() // liefert den aktuellen Datensatz
public int gibAnzahl()
public int gibPosition()
public String gibDateiname()
```

An den Rückgabe-Typen kannst du erkennen, welche Information diese Methoden dem Formular zurückgeben sollen.

Schließlich fehlen uns nur noch die Methoden, um uns in der Datenbank hin- und herzubewegen. Die Buttons vom Formular müssen also den Befehl geben können: „Gehe zum nächsten Datensatz“ oder „Springe an den Anfang der Datenbank“.

```
public void rechts()
public void links()
public void zumAnfang()
public void zumEnde()
```

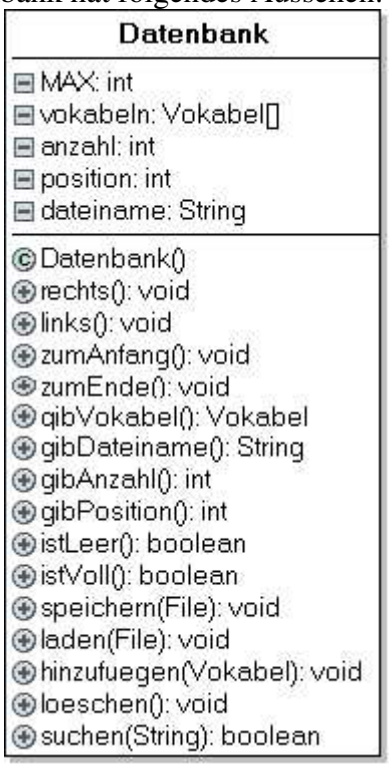
Damit haben wir alles das, was wir für unsere Datenbank benötigen. Wir wollen dieser Klasse, die unsere Daten (`vokabeln`, `dateiname`, `position` und `anzahl`) und unsere Operationen (die oben genannten 14 Methoden) zusammenfasst, den Namen `Datenbank` geben.

Aufgabe 5: Erstelle ein UML-Diagramm zur Klasse `Datenbank`. Verwende wenn möglich den Java-Editor.

Aufgabe 6: Schaue dir den vom Java-Editor erstellten Quelltext an und kommentiere die Methoden sinnvoll.

Aufgabe 7: Implementiere die Getter-Methoden der Datenbank (gibAnzahl, gibPosition, gibDateiname, gibVokabel)

Das UML-Diagramm der Datenbank hat folgendes Aussehen:



Dies ist das Grundgerüst unserer Datenbank. Hiermit wird nun ganz genau beschrieben, welche Daten und Operationen anderen Programmen zur Verfügung stehen, die diese Datenbank benutzen wollen. Du erkennst zwei zuvor nicht genannte Methoden:

```
public boolean istLeer()
public boolean istVoll()
```

Die Methode `gibAnzahl` hätte zwar gereicht, um festzustellen, ob die Datenbank leer ist, allerdings ist es üblich, für diesen Prüf-Zweck eine eigene Methode bereit zu stellen.

Aufgabe 8: Unser in Aufgabe 1 erstelltes Formular möchte diese Datenbank benutzen können. Damit dies funktioniert muss die Klasse des Formulars erweitert werden. Übernehme die unterstrichenen Änderungen in deine Formular-Klasse:

```
public class GUI extends JFrame {
  // Anfang Variablen
  private Datenbank db = new Datenbank();

  private JLabel jLabel1 = new JLabel();
  private JLabel jLabel2 = new JLabel();
```

Implementierung der Methoden

Jetzt geht es ans Eingemachte, der Programmierung der Methoden, schließlich wollen wir ja nicht nur eine leere Hülle der Datenbank haben.

Methode zum Anlegen einer neuen Datenbank – die Initialisierung

Als erstes wollen wir erreichen, dass eine neue Datenbank angelegt wird. Dies ist die Grundvoraussetzung dafür, dass man dein Programm überhaupt nutzen kann. Der Konstruktor

der Datenbank muss nur dafür sorgen, dass alle Variablen sinnvoll vorbelegt werden. Dies heißt im einzelnen:

- Die Variable `anzahl` muss auf 0 gesetzt werden (neue Datenbank ⇒ keine Datensätze).
- Die Variable `position` muss auf −1 gesetzt werden (dies ist sinnvoll, da man so anhand der Variablen Position direkt sehen kann, ob die Datenbank leer ist).
- Die Variable `dateiname` sollte auf ´NONAME.DAT´ oder ähnliches gesetzt werden.
- Die Array-Variable `vokabeln` sollte mit einem neuen Array der Länge `MAX` über den Datentyp `Vokabel` initialisiert werden.

Aufgabe 9: Implementiere den Konstruktor `public Datenbank()` der Datenbank.

```
public Datenbank() {
  anzahl = 0;
  ...
}
```

Methoden zur Erzeugung der Datensätze – das Einfügen und Löschen

Die erste Methode war noch nicht weiter schwer – kommen wir deswegen zu den Datensätzen. Die Datenbank soll über Methoden verfügen, Datensätze neu anzulegen, zu löschen, zu suchen oder zu ändern. Überlegen wir uns als erstes eine algorithmische Idee für das Anlegen von Datensätzen. Unsere Datenbank soll so angelegt sein, dass neue Vokabeln gleich sortiert (nach dem deutschen Begriff) eingefügt werden.

Aufgabe 10: Entwickle einen Algorithmus, welcher in ein Array von Vokabeln eine neue Vokabel sortiert einfügt. Orientiere dich an dem folgenden Datenbankbeispiel, in dem der Datensatz mit dem deutschen Begriff *Esel* eingefügt werden soll. Beachte auch eventuelle Spezialfälle (es gibt noch gar keine Vokabeln bzw. es ist kein Platz mehr für eine neue Vokabel).

`vokablen: Vokabel[MAX]`

1	2	3	4	5	6	7	8	9	10	11	...
Affe monkey	Bär bear	Dose tin	Eimer bucket	Fisch fish	Giraffe giraffe	Haus house	Lampe lamp	Mond moon			...

Ich hoffe, du hattest eine Idee, wie das Einfügen stattfinden könnte. Grundsätzlich gibt es natürliche mehrere Möglichkeiten, welche unterschiedlich kompliziert sind. Eine Möglichkeit ist die folgende, nur grob beschriebene:

- Wenn noch keine Vokabel existiert, dann setzte die Vokabel *Esel* an die erste Position
- Sind schon Vokabeln vorhanden, dann prüfe nach, ob überhaupt noch Platz ist.
- Ist noch Platz vorhanden, dann tue folgendes:
  - Durchlaufe das ganze Feld von vorne, bis man eine Vokabel gefunden hat, die größer als der Begriff *Esel* ist. (In unserem Beispiel würde man den Begriff *Fisch* an Position 5 feststellen.)
  - Verschiebe alle Vokabeln ab der gefundenen Position um eine Stelle nach rechts, d. h. *Mond* muss an Position 10, *Lampe* an Position 9, ... *Fisch* an Position 6.
  - Füge zum Abschluss den Begriff *Esel* an der nun freien Position ein.

Diese Möglichkeit ist in Ordnung, aber es gibt eine geschicktere:
Das Feld wird nicht von vorne nach hinten, sondern von hinten nach vorne durchlaufen, bis man eine Vokabel gefunden hat, der kleiner ist als der Begriff *Esel*.

Aufgabe 11: Versuche zu dieser Strategie ebenfalls einen Algorithmus zu entwerfen. Welche Input-, Output- und Hilfsvariablen benötigt dein Algorithmus?

Ich hoffe, du bist auf eine der folgenden Lösung ähnliche Idee gekommen:

ALGORITHMUS ***DatensatzAnlegen***

Input: vokabel: Vokabel
Global: Konstanten der Datenbank: MAX
Variablen der Datenbank: vokabeln, anzahl, position

- Falls anzahl < MAX
  dann
  - Falls anzahl = 0
    dann
    - position ← 0

    sonst
    - position ← anzahl – 2
    - solange (position >= 0) UND (vokabeln[position] >= Datensatz)
      tue
      - vokabeln[position + 1] ← vokabeln[position]
      - position ← position – 1
    - position ← position + 1
  - vokabeln[position] ← vokabel
  - anzahl ← anzahl + 1

Aufgabe 12: Implementiere die entsprechende Methode in der Datenbank-Klasse.

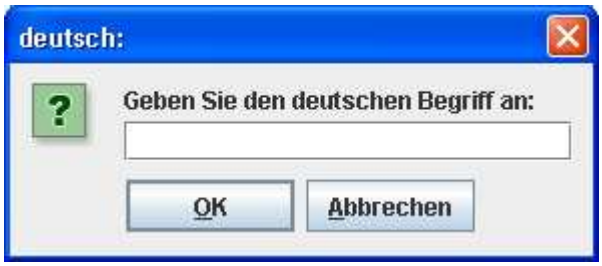
```
public void hinzufuegen(Vokabel neu) {
   ...
   // Um Vokabeln miteinander vergleichen zu können
   // benötigst du die Methode compareTo() der Klasse
   // String.
   // Z. B.:
   while (vokabeln[position - 1].gibDeutsch().
                  compareTo(neu.gibDeutsch()) > 0)
      ...
  }
```

Damit du deine eben erstellte Methode allerdings testen kannst, fehlt die entsprechende Ereignisroutine des Formulars

Wenn der Anwender auf *Datensatz|Hinzufügen* klickt, so sollte er die Möglichkeit bekommen, deutschen und englischen Begriff der neuen Vokabel einzugeben. Java bietet dir für die Benutzereingabe die Methode showInputDialog der Klasse JOptionPane:
`JOptionPane.showInputDialog(this,<Titel>,<Meldung>,JOptionPane.QUESTION_MESSAGE).`
Ruft man z. B. den folgenden Befehl auf:

```
String deutsch = JOptionPane.showInputDialog(
                    this,"Geben Sie den deutschen Begriff an:",
                    "deutsch:",JOptionPane.QUESTION_MESSAGE);
```

so erhält man das folgende Resultat:



Der Wert, den der Benutzer in das Eingabefeld eingibt, wird anschließend in der Variablen deutsch gespeichert.

Mit Hilfe dieser Methode kannst du nun nacheinander die Werte für die einzufügende Vokabel einlesen lassen:

```
public void DatensatzJMenuItem1ActionPerformed(ActionEvent evt)
{
     String deutsch = JOptionPane.showInputDialog(
                 this,"Geben Sie den deutschen Begriff an:",
                 "deutsch:",JOptionPane.QUESTION_MESSAGE);
     if (deutsch != null) {
        String englisch = JOptionPane.showInputDialog(
                 this,"Geben Sie die englische Übersetzung an:",
                 "englisch:",JOptionPane.QUESTION_MESSAGE);
        if (englisch != null) {
           Vokabel vokabel = new Vokabel(deutsch, englisch);
           db.hinzufuegen(vokabel);
           tDeutsch.setText(vokabel.gibDeutsch());
           tEnglisch.setText(vokabel.gibEnglisch());
        }
     }
  }
```

Aufgabe 13: Implementiere die Methode des Menüpunktes Datensatz|Hinzufügen. Dein Programm ist das erste mal wirklich testbereit, da jetzt Datensätze eingefügt werden können. Also, teste es aus.

Wenn man Datensätze neu anlegen kann, so sollte man auch Datensätze löschen können.

Aufgabe 14: Die Methode zum löschen des Datensatzes an der aktuellen Position ist etwas einfacher als das sortierte Einfügen eines neuen Datensatzes. Mache dir zuvor die algorithmische Idee klar und implementiere anschließend die entsprechende Methode in der Klasse Datenbank:

```
public void loeschen() {
   ...
}
```

Implementiere anschließend auch die Ereignisroutine für den Menüpunkt im Formular.

Nach dem Löschen eines Datensatzes ist irgendwie nicht so ganz klar, welcher Datensatz denn danach angezeigt werden soll. Ist es nun sinnvoll, (a) den Datensatz nach dem gelöschten oder (b) den Datensatz vor dem gelöschten Datensatz anzeigen zu lassen?

Prinzipiell stehen dir beide Möglichkeiten offen, allerdings solltest du die in beiden Fällen den jeweiligen Spezialfall beachten: Variante (a) hat den Sonderfall, dass der letzte Datensatz gelöscht wird, Variante (b) hat den Sonderfall, dass der erste Datensatz gelöscht wird.
Wenn du keine Lösung gefunden hast, kannst du im Anschluss dieses Kapitels die Prozedur zum Löschen eines Datensatzes nachlesen. Der Algorithmus benutzt die Variante (a).

Methoden zur Navigation

Beim Test hast du zwar gesehen, dass neue Datensätze auf dem Bildschirm erschienen, allerdings konnte man sich einmal eingegebene Datensätze nicht mehr anschauen. Es wäre also nötig, sich durch die Datensätze „durchzuklicken“. Schauen wir uns deshalb die vier Methoden für die Navigation an, deren erste ich dir hier vorstelle:

```
  public void rechts() {
    if (position < (anzahl - 1)) {
      position++;
    }
  }
```

[Die Abfrage `position < anzahl - 1` ist deshalb nötig, da die Position nicht auf einen noch unbelegten Datensatz zeigen sollte.]

Aufgabe 15: Implementiere genauso die drei anderen Datenbank-Methoden für die Navigation.

Diese Routinen waren ausschließlich für die Datenbank gedacht. Allerdings sollte auf dem Formular auch jedes Mal der aktuelle Datensatz angezeigt werden. Wir müssen also noch die Ereignisroutinen für die Navigations-Buttons schreiben. Der Button, mit dem man auf dem Formular den nächsten Datensatz anzeigen lässt, erhält folgende Ereignisroutine:

```
public void bVorActionPerformed(ActionEvent evt) {
    db.rechts();                                // Datenbankposition erhöhen,
    Vokabel vokabel = db.gibVokabel();          // aktuellen Datensatz holen
    tDeutsch.setText(vokabel.gibDeutsch());     // und in Textfelder
    tEnglisch.setText(vokabel.gibEnglisch()); // ausgeben.
  }
```

Aufgabe 16: Implementiere für die drei anderen Navigationsbuttons die jeweiligen Ereignisroutinen in des Formulars.

Leider gibt dein Programm im Moment noch eine Fehlermeldung aus, da die Methode gibVokabel bei einer leeren Datenbank noch kein Objekt zurückgibt. Ändere deine Methode wie folgt ab, damit die Buttons keine Fehlermeldung mehr produzieren:

```
public Vokabel gibVokabel() {
    if (position >= 0) {
      return vokabeln[position];
    } else {
      return new Vokabel("","");
    }
  }
```

Du hast wahrscheinlich gesehen, dass in allen Ereignisroutinen zur Navigation durch die Datensätze immer die gleiche Befehlesfolge auftritt:

```
Vokabel vokabel = db.gibVokabel();         // aktuellen Datensatz holen
tDeutsch.setText(vokabel.gibDeutsch());    // und in Textfelder
tEnglisch.setText(vokabel.gibEnglisch()); // ausgeben.
```

Es ist sinnvoll, sich dafür eine eigene Prozedur zu schreiben, z. B. die Methode `ausgabeAktuellerDatensatz`.

Aufgabe 17: Ändere dein Projekt wie oben angedeutet, indem du die zum Formular zugehörige Methode `ausgabeAktuellerDatensatz` implementierst und in den Navigationsbuttons verwendest.

Wenn du jetzt schon eine Prozedur zum Anzeigen des Aktuellen Datensatzes auf dem Formular erstellt hast, so solltest du diese Prozedur auch grundsätzlich verwenden. In den Ereignisroutinen des Formulars für das Einfügen und Löschen eines Datensatzes wird ebenfalls eine Anzeige des aktuellen Datensatzes nötig. Verwende also auch hier die Prozedur `ausgabeAktuellerDatensatz`.

Methoden zur Bearbeitung der Datensätze – das Ändern und Suchen

Im Zusammenhang mit der Verwaltung der Datensätze stehen jetzt noch die beiden Methoden für (a) das Suchen und (b) das Ändern eines Datensatzes aus.
Den Teil (b) überlasse ich dir alleine. Als Hinweis soll dir nur dienen, dass das Ändern eines Datensatzes damit zu bewerkstelligen ist, dass der betreffende Datensatz zuerst gelöscht wird und anschließend in geänderter Form wieder neu angelegt wird. Die Implementierung einer Methode `aendern(Vokabel vokabel)` läuft also auf die Verwendung der Methoden `loeschen()` und `hinzufuegen(...)` hinaus.

Die Frage nach dem Suchen eines Datensatzes ist da schon weitaus interessanter. Der Methodenaufruf `db.suchen("Dose")` soll z. B. bewirken, dass die Datenbank alle Datensätze nach dem Begriff „Dose“ durchsucht. Wird ein Datensatz gefunden, so sollte die aktuelle Position auf diesem Datensatz stehen bleiben. Ansonsten sollte die Position unverändert bleiben.

Die zugehörige Formular-Routine sollte folgendes machen:

```
public void DatensatzJMenuItemSuchenActionPerformed(...) {
```
- Namen eingeben lassen (Funktion showInputDialog)
- Aufruf der Datenbank-Methode suchen(...)
- Anzeige des aktuellen Datensatzes

```
};
```

Die Implementierung der drei Befehle dürfte dir nicht zu schwer fallen. Allerdings hat es die Datenbank-Routine schon etwas mehr in sich.

Aufgabe 18: Entwickle zuerst selbst eine Idee, wie das Suchen ablaufen könnte. Versuche auch, die zugehörige Methode selbst zu implementieren.

```
public boolean suchen(String deutsch) {
  ...
}
```

Wahrscheinlich durchläufst du in deinem Algorithmus die Datensätze von vorne nach hinten, bis du einen Datensatz gefunden hast, dessen Name mit dem Suchnamen übereinstimmt. Diese Art der Suche (man nennt sie die ***sequentielle Suche***) ist prinzipiell in Ordnung und würde wie folgt implementiert:

```
public boolean suchen(String deutsch) {
  int altePosition = position;
  if (anzahl >= 0) { // nur dann gibt es auch was zu suchen...
    position = 0;
    while ( position < anzahl &&
            ! vokabeln[position].gibDeutsch().equals(deutsch) )
       position++;
  }
  if (position = anzahl) position = altePosition;
}
```

Dieser Algorithmus stoppt spätestens nach dem letzten Datensatz, falls kein entsprechender Eintrag gefunden wurde. Ansonsten bleibt die Position auf dem gefundenen Datensatz stehen.

Das Suchverfahren funktioniert, allerdings kann ein Suchvorgang nach dem Begriff„Wasser“ unter Umständen ziemlich lange dauern, da er erst sehr weit am Ende der Datenbank zu finden ist.
Wenn du z. B. den Begriff„Mutter“ im Wörterbuch suchst, so fängst du wahrscheinlich auch nicht mit der ersten Seite des Buches an – du schlägst vielmehr die Mitte des Wörterbuchs auf. Das Suchverfahren funktioniert allerdings nur deswegen, weil du weißt, dass die deutschen Begriffe im Wörterbuch alphabetisch sortiert sind (zum Glück!).
Du wirst jetzt ein Suchverfahren kennen lernen, welches die Sortierung unserer Daten geschickt ausnutzt, um wesentlich schneller ans Ziel zu gelangen – die ***binäre Suche***.

ALGORITHMUS ***Binäre Suche***

Input: Suchelement: String
Output: boolean
Hilfsobjekte: links, rechts, mitte: int
Global: Konstanten der Datenbank: MAX
Variablen der Datenbank: Eintraege, Anzahl, Position

- links ← 0
- rechts ← anzahl – 1
- Solange links <= rechts:
  - mitte ← (links + rechts) / 2
  - Falls Suchelement = vokabeln[mitte].gibDeutsch()
    - dann
      - position ← mitte
      - Rückgabe true
    - sonst
      - Falls Suchelement > vokabeln[mitte].gibDeutsch()
        - dann
          - rechts ← mitte – 1
        - sonst
          - links ← mitte + 1
- Rückgabe false

Aufgabe 19: Teste das Suchverfahren mit dem Suchbegriff *Eimer* an folgender Datenbankbelegung aus (Anzahl = 10). Was passiert, wenn z. B. der Begriff *Kopf* gesucht wird?

1	2	3	4	5	6	7	8	9	10	11	...

Affe monkey	Bär bear	Dose tin	Eimer bucket	Fisch fish	Giraffe giraffe	Haus house	Lampe lamp	Mond moon			...

Aufgabe 20: Implementiere das Suchverfahren in Java. Denke an die String-Methoden compareTo(...) und equals(...)

So, nun noch wie versprochen die Routine zum Löschen eines Datensatzes:

```
public void loeschen() {
  if (anzahl > 0) {
    for (int i = position; i < (anzahl - 1); i++) {
      vokabeln[i] = vokabeln[i + 1];
    }

    vokabeln[anzahl - 1] = null;
    anzahl--;

    if (position == anzahl) {
      position--;
    }
  }
}
```

Methoden zur Dateiverwaltung – Speichern und Laden

So weit so gut, allerdings müsste man bei dem aktuellen Stand des Projekts bei jedem Programmstart sämtliche Daten neu eingeben. Besser wäre es da, wenn man alle Daten gesammelt in eine Datei auf der Festplatte (oder andere Datenträger) speichern könnte. Und genau darum soll es in diesem Kapitel gehen.

Das **Speichern und Laden** einer Datei innerhalb eines eigenen Projektes erfolgt am besten über die beiden von Java schon zur Verfügung gestellten Dialoge **JFileChooser.showOpenDialog()** und **...showSaveDialog()**. Du findest diese beiden Oberflächenobjekte in der Werkzeugleiste **Swing 2**.

Die Dialoge „OpenDialog“ und "SaveDialog"

Verwende die Methode showOpenDialog(this), um einen Windows-Standarddialog zum Öffnen von Dateien zu erzeugen, mit dessen Hilfe die Benutzer den Namen einer zu öffnenden Datei eingeben können. Die vom Benutzer im Dialogfenster ausgewählte Datei wird mithilfe der Methode *getSelectedFile* ausgelesen. Möchtest du nur bestimmte Dateiendungen im Dialog angezeigt bekommen, so kannst du dir eigene FileFilter erstellen, welche dem JFileChooser-Objekt hinzugefügt werden, allerdings wollen wir das hier nicht weiter thematisieren.
Analog zur Methode showOpenDialog() kannst du die Methode showSaveDialog verwenden.

Aufgabe 21: Füge beide Dialoge deinem Projekt hinzu..

Klickt der Benutzer in deinem Projekt z. B. auf den Menüpunkt Datei öffnen, so solltest du in der zugehörigen Ereignisroutine des Formulars folgende Programmzeilen haben:

```
public void DateiJMenuItemSpeichernActionPerformed(ActionEvent evt) {
  JFileChooser f = new JFileChooser();
```

```
  f.setSelectedFile(new File(db.gibDateiname()));
  int state = f.showSaveDialog(this);               // Save-Dialog starten
  if (state == f.APPROVE_OPTION) {                  // Wenn OK gedrückt
    File file = f.getSelectedFile();
    db.speichern(file);
  }
}
```

Wie du siehst, ist die Methode ***showSaveDialog()*** so realisiert, dass im Funktionsergebnis festgehalten wird, wie der Anwender den Dialog beendet hat. Bei Auswahl des OK-Buttons innerhalb des Dialogfensters liefert die Funktion den Wert *JFileChooser.APPROVE_OPTION*. In diesem Fall wird die Datenbank-Methode `db.speichern(...)` aufgerufen, um welche wir uns jetzt im Anschluss kümmern müssen.

Aufgabe 22: Erweitere dein Projekt um die Formular-Ereignisroutinen zum Speichern und Laden einer Datei.

Die Methoden der Datenbank

Die Verwendung der Dialoge war etwas knifflig, dafür wird das Speichern der Datenbank etwas einfacher, da die zugehörige Algorithmik im wesentlichen mit der der *sequentiellen Suche* übereinstimmt.
Java stellt uns die Klasse FileWriter zur Verfügung. Bei der Erzeugung eines Objekts dieser Klasse können wir den File (aus dem FileChooser-Objekt) mitgeben, so dass in diese Datei geschrieben wird. Die so entstehende Datei können wir später auch in einem Texteditor ansehen und bearbeiten, es handelt sich also um eine reine Textdatei.
Um alle Vokabeln zu speichern müssen wir nun nur noch eine Schleife durchlaufen, in der wir alle Vokabeln in deutscher und englischer Bedeutung in die Datei schreiben.
Zum Abschluss muss die Datei noch geschlossen werden, damit das Betriebssystem die Datei wieder zur Bearbeitung frei geben kann.
Insgesamt ergibt sich der folgende Programmtext für die Methode speichern:

```
public void speichern(File datei) {
  try {
    FileWriter writer = new FileWriter(datei);
    for (int i = 0; i < anzahl; i++) {
      String s = vokabeln[i].gibDeutsch() + ";" +
                 vokabeln[i].gibEnglisch();
      writer.write(s + "\n");
    }
    writer.close();
  } catch (IOException e) {
    System.err.println(e.toString());
  }
}
```

Der try-catch-Block dient dazu, einen eventuell auftretenden Fehler abzufangen. Es könnte z. B. sein, dass die gewählte Datei keine Schreibrechte zulässt oder dass der ausgewählte Dateiname fehlerhaft ist. Die entsprechende Fehlermeldung würde auf der Konsole ausgegeben werden.

Aufgabe 23: Implementiere die Speichern-Methode in deiner Datenbank-Klasse. Führe dein Projekt anschließend aus, füge einige Datensätze hinzu und speichere die Datenbank in einer neuen Datei. Öffne anschließend die Datei mit einem Texteditor und überprüfe die Datensätze.

So einfach, wie das Speichern über einen FileWriter ist, so schwierig ist das Öffnen der gleichen Datei, um die Datensätze wieder einzulesen. Schaue dir zuerst den Quellcode genau an:

```
public void laden(File datei) {
  try {
    FileReader reader = new FileReader(datei);
    BufferedReader puffer = new BufferedReader(reader);
    String s = puffer.readLine();
    anzahl = 0;

    while (s != null) {
      String deutsch = s.substring(0, s.indexOf(";"));
      String englisch = s.substring(s.indexOf(";") + 1);
      Vokabel neu = new Vokabel(deutsch, englisch);
      vokabeln[anzahl] = neu;
      anzahl++;
      s = puffer.readLine();
    }

    position = 0;
    reader.close();
  } catch (FileNotFoundException e) {
    System.out.println("Datei nicht vorhanden");
  } catch (IOException e) {
    System.err.println(e.toString());
  }
}
```

In Java gibt es genauso wie den FileWriter auch einen FileReader. Allerdings liefert dieser eine Textdatei nur Zeichenweise – wir benötigen die Datei aber Zeilenweise. Deshalb benötigen wir einen BufferedReader, welcher stets eine ganze Zeile (readLine()) aus der Textdatei auslesen kann.
Der Hauptteil der Methode beschäftigt sich nun mit dem Auslesen der einzelnen Zeilen: Solange noch Zeilen in der Textdatei gespeichert sind (s != null) werden diese aus der Datei ausgelesen und in deutsche und englische Begriffe zerlegt (s.substring(...)). Mit jeder gefundenen Zeile wird die Anzahl um eins erhöht und die nächste Zeile aus der Datei ausgelesen.
Zum Abschluss wird noch die aktuelle Position auf den Anfang der Vokabelliste gesetzt und die Datei geschlossen.
Auch hier gibt es wieder einen try-catch-Block, um eventuelle Dateifehler abzufangen.

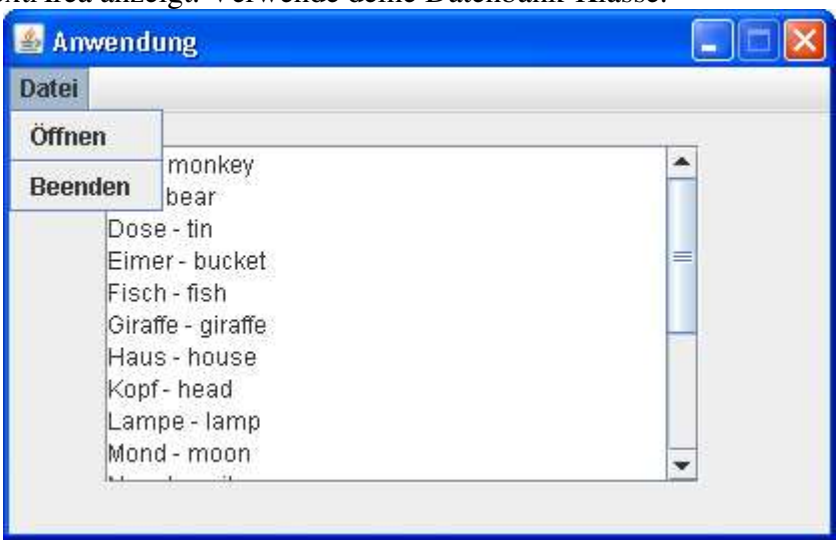
Aufgabe 24: Implementiere die laden-Methode in deiner Datenbank-Klasse und überprüfe, ob du die zuvor gespeicherten Datensätze wieder einlesen kannst.

So, jetzt müsste deine Datenbank eigentlich gut funktionieren. Man sollte nun in deinem Projekt eine Vokabel-Datenbank anlegen können, man sollte die Datenbank speichern und öffnen können. Die Bearbeitung der Datensätze umfasst das neue Anlegen, Löschen, Ändern und Suchen von Datensätzen. Perfekt.

Anwendung der Datenbank in anderen Projekten

Wir wollen nun testen, ob wir unsere Datenbank-Klasse auch in anderem Kontext verwenden können. Dazu soll ein neues Projekt erstellt werden, welches lediglich lesenden Zugriff auf die Datenbank erhalten soll, eine sogenannte neue Sicht (View) für unsere Datenbank.

Aufgabe 25: Erstelle ein ganz neues Projekt, in dem der Benutzer eine bestehende Datenbank von der Festplatte öffnen kann und welches ihm daraufhin alle Datensätze in einer TextArea anzeigt. Verwende deine Datenbank-Klasse.



Aufgabe 26: Die Datensätze sollen neben dem deutschen und englischen Begriff auch noch zwei mögliche andere englische Bedeutungen enthalten. Ändere die Datenbank entsprechend ab.

Du hast in den letzten beiden Aufgaben gesehen, wie schnell nun deine bestehende Datenbank erweitert werden kann und wie einfach sich diese in andere Programme einbinden lässt. Ein wesentlichen Nachteil hat die ganze Sache aber noch. Die maximale Anzahl der Datensätze ist von vorneherein festgelegt. Zwar könntest du die Konstante `MAX` erhöhen, aber irgendwann ist Schluss. Ärgerlich ist außerdem, dass man sich zu Anfang festlegen muss, wie viele Datensätze maximal zugelassen werden. Will man diese Maximalanzahl ändern, so muss man den Quellcode deines Programms ändern. Allerdings ist es nicht üblich, Programme als Quellcode zu verkaufen – oder hast du den Quellcode von Microsoft Word? – also muss es eine andere Möglichkeit geben, mit der die Anzahl der Datensätze nicht festgelegt werden muss.
Die Festplatte hat ein wesentlich größeres Speichervolumen als der RAM. Außerdem hast du beim Speichern der Datenbank gesehen, dass die angelegte Datei nur so viele Datensätze umfasste, wie wirklich benötigt wurden. Das ARRAY dagegen hatte eine Menge ungenutzter Datensätze. Also, warum sollten wir unsere Daten nicht sofort auf der Festplatte verwalten? Microsoft Word macht es genauso, denn das Programm weiß ja auch nicht von vorneherein, wie viele Seiten du mit dem Programm schreiben willst. Also legt sich Word eine Datei auf der Festplatte an und sämtliche Änderungen des Textes werden auf dieser Datei durchgeführt.

Projekt „Minidatenbank“ – externe Datenverwaltung

Ab hier ist das Skript noch an DELPHI angepasst. Ich weiß noch nicht, wann ich das Skript ändere.

Wenn die Daten der Datenbank nicht mehr intern in einem ARRAY, sondern extern in einer Datei verwaltet werden, so muss sich zwangsläufig die Typdeklaration der Datenbank ändern:

```
unit U_Dat_ex;

interface

{ Eine Konstantendeklaration benötigen wir nicht mehr. }

type  TEintrag  = record
                    Name, Telefon: string;
                  end;
      TEintraege= File of TEintrag;
      TDatenbank= class
                                      { DATEN der Klasse TDatenbank }
        Eintraege: TEintraege;
        Position: integer;
        Anzahl: integer;
        DateiGeoeffnet: boolean;
        Dateiname: string;
                                      { METHODEN der Klasse TDatenbank }
... { wie vorher }
```

Es gibt in der Typ-Deklaration zwei wesentliche Unterschiede zur ersten Version:

- Es gibt keine Beschränkung der maximalen Anzahl an Datensätzen, da die Variable `Eintraege` eine *Filevariable* ist. Deshalb fällt die Konstantendeklaration weg.
- Das Flag DateiGeaendert ist überflüssig, da alle Veränderungen der Datenbank in jedem Fall auf der Festplatte vorgenommen werden. Ein versehentliches Verwerfen der Änderungen an der Datenbank ist damit ausgeschlossen. Dafür muss aber sichergestellt sein, dass eine Datei geöffnet vorliegt. Ansonsten sind Änderungen an der Datenbank nicht möglich. Deshalb wird ein Flag `DateiGeoeffnet` als boolesche Variable eingefügt.

Die Datenbankoperationen bleiben algorithmisch gesehen gleich. Allerdings kann man nun nicht mehr so einfach auf einen einzelnen Datensatz zugreifen, wie bisher, denn öffnet man eine Datei, so steht die *Filemarke* auf dem ersten Datensatz und nicht auf dem Datensatz, welcher durch die Variable `Position` markiert ist. DELPHI bietet dir allerdings einen Befehl, mit dem du die *Filemarke* auf einen beliebigen Datensatz setzen kannst:

`Seek(<Dateivariable>, <Positionsindex>)`

Die Verwendung dieses Befehls wird am deutlichsten, wenn du dir die beiden Datenbankoperationen DatensatzLesen sowohl in der ersten Fassung als auch in der Fassung mit externer Datenverwaltung anschaust:

Zur Erinnerung, dies war die alte Fassung:

```
procedure TDatenbank.DatensatzLesen(var date: TEintrag);
begin
  if (Position = 0)
    then begin
           date.Name:= ´´;
           date.Telefon:= ´´;
         end
    else date:= Eintraege[Position];
end;
```

In der Datenbank-Unit mit externer Speicherung müsste diese Routine wie folgt formuliert werden:

```
procedure TDatenbank.DatensatzLesen(var date: TEintrag);
begin
  if (Position = 0) or (not DateiGeoeffnet)
    then begin
           date.Name:= ´´;
           date.Telefon:= ´´;
         end
    else begin
           Seek(Eintraege, Position - 1)
           Read(Eintraege, date);
         end;
end;
```

Vielleicht wirst du jetzt stutzig, dass im Seek-Befehl auf Position – 1 zugegriffen wird? Die Datensätze in Dateien sind mit 0 beginnend durchnummeriert, nicht mit 1 beginnend wie es im ARRAY der Fall war.

Weitere nützliche Befehle im Zusammenhang mit Dateien sind die folgenden:

`FilePos(<Dateivariable>)`

ist eine Funktion, welche dir die Nummer des Datensatzes ausgibt, auf der die *Filemarke* aktuell steht. Der Befehl `Seek(Eintraege, FilePos(Eintraege)−1)` bewirkt also, dass die *Filemarke* um eine Stelle zurückgesetzt wird.

`FileSize(<Dateivariable>)`

ist eine Funktion, welche dir die Anzahl der gespeicherten Datensätze ausgibt. Der Befehl `Seek(Eintraege, FileSize(Eintraege))` bewirkt also, dass die *Filemarke* hinter den letzten Datensatz gesetzt wird. Dies kann sehr nützlich beim Anfügen eines neuen Datensatzes sein.

`Erase(<Dateivariable>)`

ist eine Prozedur, welche die momentane Datei von der Festplatte löscht.

Truncate(<Dateivariable>)

ist eine Prozedur, welche alle Datensätze ab dem durch die *Filemarke* markierten Datensatz abschneidet. Diese Prozedur benötigst du, wenn ein Datensatz gelöscht wird.

Rename(<Dateivariable>,<Dateiname>)

ist eine Prozedur, welche der aktuellen Datei einen neuen Dateiname verpasst.

Mit diesen Befehlen müsste es dir eigentlich gelingen, die Datenbank-Unit auf die externe Verwaltung von Datensätzen umzustellen. Am einfachsten ist es, wenn du die bereits bestehende Datenbank-Unit unter einem anderen Namen abspeicherst und dann nur noch die nötigen Änderungen vornimmst.

Aufgabe 24: Implementiere die Unit `U_Dat_ex.pas`, mit der eine Datenbank extern verwaltet wird. Da sich die Namen der Datenbankoperationen nicht geändert haben, sollte dein Projekt auch mit dieser Unit funktionieren. Probiere es aus.

```
Unit Unit1;                                    {Formularunit }
uses SysUtils, WinTypes, ..., U_Dat_ex;
       ...
var Form1: TForm1;
    DB: TDatenbank;
...
```

Aufgabe 25: Teste aus, ob auch das Programm aus Aufgabe 22 mit dieser Unit klar kommt.