Brandenburgische
Technische Universität
Cottbus - Senftenberg

Brandenburgische Technische Universität Cottbus – Senftenberg
Institut für Informatik
PD Dr.-Ing. habil. Thomas Hinze

Sommersemester 2016

Abgabe der Lösungen: bis Mittwoch, 13.07.2016

Einführung in die Programmierung
Übungsblatt 6

Hinweise

- Erstellen Sie bitte eine schriftliche Lösung der nachfolgenden Aufgaben und erzeugen Sie daraus eine pdf-Datei. Sie können auch Ihre (lesbare!) handschriftliche Lösung als pdf einscannen. Achten Sie bitte darauf, dass am Beginn der ersten Seite des pdfs Ihr *Name*, Ihre *Matrikelnummer* und der Name Ihres *Tutors* (Laborübungsgruppe) vermerkt ist. Jede Aufgabe, die durch **(Laborübung)** gekennzeichnet ist, beinhaltet als Lösung auch ein Java-Programm als Quelltextdatei. Der Quelltextdateiname muss dem Namen der *Klasse* entsprechen, die die `main`-Methode enthält. Bitte packen Sie Ihr pdf gemeinsam mit allen Java-Quelltextdateien (nur `*.java`, keine `*.class`) in eine zip-Datei, die Sie termingemäß im moodle hochladen.
- Sie bekommen eine Bewertung Ihrer Lösung mit der Information, ob sie ausreichend zum Bestehen des Übungsblattes ist. Im Falle einer nichtausreichenden Lösung haben Sie Gelegenheit, eine verbesserte Version zeitnah, aber spätestens bis 22.07.2016, im moodle nachzureichen. Plagiate werden nicht als ausreichende Lösungen anerkannt.

Aufgabe 1

(Laborübung) In natürlichsprachigen Texten variiert die Häufigkeit, mit der die einzelnen Buchstaben vorkommen. Beispielsweise enthalten deutsche Texte üblicherweise deutlich mehr Buchstaben *e* als *x*. Die prozentuale Verteilung der Buchstabenhäufigkeiten lässt sich als „Fingerabdruck“ der entsprechenden natürlichen Sprache auffassen. Für zahlreiche Sprachen hat man diese Häufigkeitsverteilung erstellt, indem eine Vielzahl von Texten aus dem Literaturschaffen, aber auch Alltagsjournalismus, Fachbücher und Schriftensammlungen akribisch ausgewertet wurden. In der *englischen* und in der *deutschen* Sprache ergibt sich daraus jeweils folgende prozentuale Verteilung, wobei die deutschen Umlaute *ä* durch *ae*, *ö* durch *oe*, *ü* durch *ue* sowie das Sonderzeichen *ß* durch *ss* ersetzt wurden. Groß- und Kleinschreibung wird nicht unterschieden:

	a	b	c	d	e	f	g	h	i	j	k	l	m
englisch	8.167	1.492	2.782	4.253	12.702	2.228	2.015	6.094	6.966	0.153	0.772	4.025	2.406
deutsch	6.51	1.89	3.06	5.08	17.40	1.66	3.01	4.76	7.55	0.27	1.21	3.44	2.53

	n	o	p	q	r	s	t	u	v	w	x	y	z
englisch	6.749	7.507	1.929	0.095	5.987	6.327	9.056	2.758	1.628	0.114	0.387	0.308	0.136
deutsch	9.78	2.51	0.79	0.02	7.00	7.58	6.15	4.35	0.67	1.89	0.03	0.04	1.13

Es zeigt sich, dass die Buchstabenhäufigkeiten im Deutschen etwas stärker streuen als im Englischen.

Schreiben Sie ein Java-Programm, das eine Textdatei einliest und die Häufigkeiten der darin enthaltenen Buchstaben ermittelt. Es empfiehlt sich, den Inhalt der Textdatei in eine einzige Zeichenkette abzulegen. Liegt der auszuwertende Dateiinhalt in einer Zeichenkette vor, werden alle enthaltenen Großbuchstaben in

Kleinbuchstaben umgewandelt. Dazu steht auf Zeichenketten die Methode `toLowerCase` zur Verfügung. Anschließend werden die Buchstabenhäufigkeiten ausgezählt und dazu in einem gesonderten Feld abgelegt. Die Kleinbuchstaben decken als Unicode-Werte den Bereich von 97 (`'a'`) bis 122 (`'z'`) ab. Zusätzlich wird die Gesamtzahl Buchstaben bestimmt, so dass die prozentuale (relative) Häufigkeit jedes Buchstaben von `a` bis `z` berechnet werden kann. Seien `h['a']` bis `h['z']` die aus der eingelesenen Textdatei gewonnenen prozentualen Buchstabenhäufigkeiten. Ihr Programm soll diese Häufigkeitsverteilung ausgeben und abschließend die aufsummierte quadratische Abweichung sowohl zur Buchstabenverteilung in der englischen Sprache als auch zur Buchstabenverteilung in der deutschen Sprache unter Nutzung der obigen Tabellen bestimmen:

$$\texttt{abweichung\_englisch} = \sum_{\texttt{i='a'}}^{\texttt{'z'}} (\texttt{h[i]} - \texttt{englisch[i]}) \cdot (\texttt{h[i]} - \texttt{englisch[i]})$$

$$\texttt{abweichung\_deutsch} = \sum_{\texttt{i='a'}}^{\texttt{'z'}} (\texttt{h[i]} - \texttt{deutsch[i]}) \cdot (\texttt{h[i]} - \texttt{deutsch[i]})$$

Die Sprache mit der kleinsten aufsummierten quadratischen Abweichung ist dann die vom Programm prognostizierte Sprache des Textes. Testen Sie Ihr Programm anhand der Textdateien in `textsammlung.zip` aus. Dort enthalten ist die bissige Informatiker-Satire *Bastard Operator* sowie das Märchen *Hänsel und Gretel* jeweils in einer inhaltsgleichen deutschen und englischen Fassung. Zusätzlich steht die Datei `lorem-ipsum.txt` bereit, deren Inhalt weder deutsch noch englisch ist (sondern Pseudo-Latein). Protokollieren Sie bitte für alle Textdateien die aufsummierten quadratischen Abweichungen der Buchstabenhäufigkeiten zum Deutschen und zum Englischen und überzeugen Sie sich, dass die vom Programm vorgenommene Sprachenzuordnung (bis auf *lorem ipsum*) mit der tatsächlichen Sprache des jeweiligen Textes übereinstimmt.

Aufgabe 2

Lineare Listen bilden eine leicht programmierbare *dynamische Datenstruktur*, die sich zur Datenhaltung in zahlreichen Anwendungsszenarien vorteilhaft nutzen lässt. Insbesondere kleine Datenbanken, bei denen häufig Datensätze hinzugefügt oder gelöscht werden, profitieren von der Flexibilität linearer Listen. Im Gegensatz zum Feld, dessen Größe (Anzahl Feldelemente) sich nach dem Anlegen nicht mehr verändern kann, lassen sich lineare Listen während der Programmlaufzeit verkürzen oder verlängern. Ihr Speicherplatzbedarf passt sich somit gleitend den Erfordernissen der Datenbanknutzung an. Jeder Datensatz einer linearen Liste stellt ein *Listenelement* dar. Neben den Nutzerdaten enthält jedes Listenelement in einer *einfach verketteten* linearen Liste zusätzlich einen Verweis (Referenz) auf das unmittelbar nachfolgende Listenelement. Beim letzten Element der Liste wird stattdessen jedoch der Vermerk `null` eingetragen, wodurch das Ende der Liste markiert ist. Für die praktische Arbeit mit einer linearen Liste legt man einen globalen Verweis (Anker) auf das führende Listenelement an. Das Durchlaufen einer einfach verketteten linearen Liste erfolgt elementweise vom Anker aus. Während der Laufzeit des listenverwaltenden Programms können neue Listenelemente angelegt sowie bestehende Listenelemente gelöscht werden. Der Speicherbedarf der Liste variiert folglich bei der Programmabarbeitung und ist mithin dynamisch.

(Laborübung) Mithilfe einer einfach verketteten linearen Liste sollen die Ergebnisse einer Meisterschaft oder Spielsaison einer Mannschafts-Ballsportart (z.B. Fußballmeisterschaft) mit mehreren Spielen erfasst und ausgewertet werden. Ein Datensatz (Listenelement) besitzt dabei folgende Struktur:

```
class Spiel {
  String mannschaft1;
  String mannschaft2;
  int tore1;
  int tore2;
  Spiel next; //Verweis auf nachfolgendes Listenelement

  public Spiel(String mannschaft1, String mannschaft2, int tore1, int tore2, Spiel next)
  {
    this.mannschaft1 = mannschaft1;
    this.mannschaft2 = mannschaft2;
    this.tore1 = tore1;
    this.tore2 = tore2;
    this.next = next;
  }
}
```

Im Begleitmaterial finden Sie in der Datei `MeinMeisterschaftsManager.java` die Klasse `Spiel` sowie den Lückenquelltext der Klasse `Meisterschaft`, der einige nur als Dummies eingetragene Methoden enthält. Ihre Aufgabe besteht darin, die Implementierungen dieser Methoden sowie die `main`-Methode der Klasse `MeinMeisterschaftsManager` zu vervollständigen (jeweils unter der Kommentarzeile `//Bitte hier Quelltext eintragen`):

- Die Methode `fuegeSpielHinzu(String m1, String m2, int tore1, int tore2)` fügt ein neues Spiel mit den übergebenen Parametern am *Anfang* der Liste ein.
- Die Methode `anzahlSpiele()` liefert die Anzahl der in der gesamten Liste erfassten Spiele zurück.
- Die Methoden `gibMannschaft1(int index)` bis `gibTore2(int index)` liefern zum Spiel (Listenelement) an der Position `index` die entsprechend hinterlegten Daten.
- Die Methode `zeigeSpiellisteAn()` gibt die gesamte Spielliste in tabellarischer Form am Bildschirm aus.
- Die Methode `gesamtzahlTore()` ermittelt die Gesamtzahl Tore über alle erfassten Spiele.
- **(optional)** Die Methode `loescheSpiel(int index)` löscht das Spiel (Listenelement) an der Position `index` aus der Liste. Das Listenelement am Listenanfang hat dabei den Index `0`, das letzte Listenelement den Index `anzahlSpiele() - 1`.

In der `main`-Methode wird ein Objekt Ihrer Klasse `Meisterschaft` angelegt und verwendet. Fügen Sie unter Nutzung der Methode `fuegeSpielHinzu` die Ergebnisse einer Meisterschaft oder Spielsaison (mindestens 20 Spiele) in einer Mannschafts-Ballsportart Ihrer Wahl ein (z.B. Fußball, Handball, Volleyball, Basketball, Football, Eishockey, Wasserball, ...). Bei internationalen Meisterschaften können Sie für die Mannschaftsnamen entsprechende Länderabkürzungen verwenden wie z.B. `DE` für Deutschland oder `BR` für Brasilien. Mittels der vorimplementierten Methode `mannschaftenMaxTore` können Sie die Mannschaften absteigend nach der Anzahl erzielter Tore anzeigen lassen sowie darüber hinaus mit den entsprechenden Methoden die Spielliste, die Gesamtzahl Spiele und die Gesamtzahl Tore ausgeben.

Es bieten sich die Daten der *Fußball-Europameisterschaft 2016* an (`https://de.wikipedia.org/wiki/Fu%C3%9Fball-Europameisterschaft_2016`) mit insgesamt 24 beteiligten Ländern und 51 Spielen ebenso wie die *Fußball-Weltmeisterschaft 2014* (`https://de.wikipedia.org/wiki/Fu%C3%9Fball-Weltmeisterschaft_2014`) mit 32 Ländermannschaften und 64 Spielen.