- *Debug Debug*

Inhaltsverzeichnis

Dokumentation

- Java API
- Javainsel
- javabuch.de

Status

Noch 0 Minuten
Weg damit, das will
ich gar nicht wissen!

Alles Speichern

Objekte und Datenstrukturen

Implementieren Sie alle Aufgaben in den dafür vorgesehenen Dateien im Ordner "impl".

- Bitte lesen Sie die Angabe **komplett** durch, bevor Sie mit der Umsetzung beginnen.
- Halten Sie sich genau an die Angabe und geben Sie nur die geforderten Daten aus.
- Haben Sie **Schwierigkeiten** bei der Umsetzung einer der Teilaufgaben, **versuchen Sie andere Teile zuerst umzusetzen** und wenden Sie sich zuletzt nochmal der ungelösten Teilaufgabe zu. Sie erhalten auch Punkte für nicht vollständig implementierte Teilaufgaben

Programmieraufgabe

(30 Punkte)

Implementieren Sie Funktionen, die einfache Manipulationen und Auswertungen eines Textes ermöglichen.

Aufgabenstellung

Die Aufgabe umfasst die Implementierung der geforderten → Methoden in den angegeben → Klassen.

Funktionale Anforderungen

Ein Objekt einer Klasse `Text` repräsentiert einen mehrzeiligen Text. Dieser wird wiederum als eine Liste von Zeilen-Objekten dargestellt. *Sie können davon ausgehen, dass nur Texte vorkommen, die aus mindestens einer Zeile bestehen und keine Leerzeilen enthalten.*

Eine `Zeile` wird durch ein Objekt der Klasse Line dargestellt. Sie wird als eine Liste von Wörtern gespeichert. Ein Wort besteht aus *mindestens einem Zeichen* und wird als String-Objekt dargestellt. Es kommen also *keine Leerstrings* vor. Sie können davon ausgehen, dass nur Zeilen vorkommen, *die mindestens ein Wort enthalten*.

Erstellen Sie die Klassen Text und Line. Wählen Sie als Instanzvariable geeignete Collection-Objekte zur internen Repräsentation eines Textes bzw. einer Zeile. Diese Instanzvariablen sollen von außen nicht sichtbar sein.

Ergänzen Sie in der main-Methode das Einlesen des Texts um den Aufbau der beschriebenen Repräsentation (Text-Objekt und Line-Objekte) und testen Sie ihre Methodenimplementierungen mittels der entsprechend in der main-Methode enthaltenen Aufrufe.

Nicht-funktionale Anforderungen

Nutzen Sie wenn möglich immer bestehenden Code und **vermeiden Sie so doppelte Codestücke**. Sie können bei Bedarf auch zusätzliche Methoden einführen. Dies sollten Sie jedoch nur dann machen, wenn es notwendig ist oder im Rahmen von **Codewiederverwendung** Sinn macht. Achten Sie bei Ihren Klassen auf **korrekte Datenkapselung**, also insbesondere auf die richtigen (Sichtbarkeits-)Modifikatoren bei Methoden und Datenfeldern.

Herangehensweise

Implementierungsreihenfolge

Folgende Vorgehensweise ist bei der Umsetzung der Aufgabe empfehlenswert:

- Definieren Sie die Instanzvariable(n) der Klasse Line. Wählen Sie eine geeignete Collection-Klasse, die eine Liste von Strings repräsentiert.
- Implementieren Sie die Methoden von Line.
- Definieren Sie die Instanzvariablen(n) der Klasse Text. Wählen Sie eine geeignete Collection-Klasse, die eine Liste von Zeilen (Line-Objekten) repräsentiert, und eine geeignete Collection-Klasse die die zahlenmäßigen Vorkommnisse der einzelnen Wörter repräsentiert.
- Implementieren Sie die Methoden von Text.
- Zur Implementierung von numberOfWords der Klasse Text können Sie numberOfWords der Klasse Line verwenden.
- Zur Implementierung von substitute der Klasse Text können Sie substitute der Klasse Line verwenden.
- Mittels der Methode getWordIterator kann auf den ganzen Text so zugegriffen werden, als stünde er in einer einzigen Zeile. Damit können Sie Methoden implementieren, für die die Zeilenstruktur keine Rolle spielt (calculateWordCounts). In der Implementierung von getWordIterator erzeugen Sie zuerst diese eine (lange) Zeile und geben deren Iterator zurück.

Testen

Kompilieren Sie früh und oft und versuchen Sie Fehler sofort zu beheben. Testen Sie Ihre fertige Implementierung unter Nutzung des mitgelieferten Ein- und Ausgabe-Paares im Ordner `io`. Rufen Sie Ihr Programm dafür wie folgt auf: `java TextMan < ../io/spec.i01`

Vergleichen Sie die Ausgabe des Programms anschließend mit der geforderten Ausgabe in den entsprechenden Dateien (z.B.: `spec.o01`). Wenn Sie direkt auf der Konsole Eingaben vornehmen, beenden Sie die Eingabe mit Strg+D.

Klassen und Methoden

Die folgenden Klassen und Methoden sind zu implementieren. Falls nicht anders angegeben, können Sie davon ausgehen, dass die den Methoden bei einem Aufruf übergebenen Werte gültig sind (daher beispielsweise keine `IndexOutOfBoundsException` verursachen). Sie müssen die Gültigkeit daher nicht überprüfen.

`TextMan`
: Diese Klasse ist ausführbar und beinhaltet daher die `main`-Methode.
`public static void main(String[] args)`

  Ergänzen Sie das Einlesen des Textes um den Aufbau der Repräsentation mittels der Klassen Text und Line an den entsprechenden Stellen.

`Text`
: Ein Text-Objekt repräsentiert einen mehrzeiligen Text. Dieser wird als eine Liste von Zeilen-Objekten dargestellt. Es kommen nur Texte vor, die aus mindestens einer Zeile bestehen und keine Leerzeilen enthalten.
`void addLine(Line line)`

  Fügt eine Zeile am Ende des Texts an.

`Iterator getLineIterator()`

  Liefert einen Iterator über alle Zeilen des Texts

`int numberOfLines()`

  Liefert Anzahl der Zeilen

`Iterator getWordIterator()`

  Liefert einen Iterator über alle Wörter des Texts

`int numberOfWords()`

  Liefert Anzahl der Wörter. Tipp: Benutzen Sie getWordIterator()

`void calculateWordCounts()`

  Berechnet für jedes im Text auftretende Wort die Anzahl seiner Vorkommnisse und speichert Sie in einer geeigneten Instanzvariablen

`HashMap getWordCounts()`

Liefert die zuletzt berechneten (mittels calculateWordCounts) Wortvorkommnisse

`void substitute(String word, String newWord)`

Ersetzt ein Wort im ganzen Text durch ein anderes. Tipp: Verwenden Sie dabei die Methode substitute der Klasse Line

`Line`

Ein Line-Objekt repräsentiert eine Zeile. Diese wird als eine Liste von Wörtern dargestellt. Ein Wort besteht aus mindestens einem Zeichen und wird als String-Objekt dargestellt. Es kommen keine Leerstrings vor. Es kommen nur Zeilen vor, die mindestens ein Wort enthalten.

`void addWord(String word)`

Fügt ein Wort am Ende einer Zeile an.

`Iterator getIterator()`

Liefert einen Iterator über alle Wörter der Zeile

`int numberOfWords()`

Liefert Anzahl der Wörter

`void substitute(String word, String newWord)`

Ersetzt ein Wort in der Zeile durch ein anderes

`public String toString()`

Liefert einen formatierten String mit allen Wörtern der Zeile: vor jedem Wort steht ein '+'-Zeichen, zwischen den Wörtern keine Leerzeichen

Testfälle

Kompilieren Sie früh und oft und versuchen Sie Fehler sofort zu beheben. Testen Sie nach jedem Kompilieren Ihre Implementierung. Wenn Sie direkt auf der Konsole Eingaben vornehmen, beenden Sie die Eingabe mit Strg+D.

Automatisiertes Testen

Um alle vorhandenen Testfälle auf einmal auszuführen und so Ihre Implementierung ausführlich zu prüfen, können Sie mit dem aus der Übung bekannten Skript `batchrun` automatisch alle Testfälle ausführen. Führen Sie dazu im Terminal im Testverzeichnis folgenden Befehl aus: `batchrun`
Dieses Skript nutzt, so vorhanden, auch weitere Testfälle aus dem Verzeichnis `batch`.

spec.$1

INPUT

```
heute       regnet es und morgen vielleicht auch
aber gestern nicht


vielleicht     ist es morgen sonnig und     es regnet vielleicht
         auch
```

OUTPUT

```
 4 lines. 20 words.
 +heute+regnet+es+und+morgen+vielleicht+auch
 +aber+gestern+nicht
 +vielleicht+ist+es+morgen+sonnig+und+es+regnet+vielleicht
 +auch
 auch 2x
 und 2x
 morgen 2x
 nicht 1x
 vielleicht 3x
 regnet 2x
 sonnig 1x
 heute 1x
 ist 1x
 gestern 1x
 aber 1x
 es 3x
```

```
    +heute+regnet+es+und+morgen+sicher+auch
    +aber+gestern+nicht
    +sicher+ist+es+morgen+sonnig+und+es+regnet+sicher
    +auch
```