Bereich: Input/Output (3)

Zugriff auf eine Textdatei	Schwierigkeit: ★★☆☆☆
Package: `de.dhbwka.java.exercise.io.textfile`	**Klasse:** `TextFile`

Aufgabenstellung:

Schreiben Sie eine Klasse `TextFile`, welche eine Textdatei verwalten soll. Diese soll folgende Methoden haben:

- Konstruktor **`TextFile`**`(File f)`
  zum Erzeugen eines neuen `TextFile`-Objektes. Der Konstruktor soll die entsprechende Datei sofort einlesen. (Die zugehörige Datei muss bereits existieren.)
- Konstruktor **`TextFile`**`(String pathname)`
  zum Erzeugen eines neuen `TextFile`-Objektes. Der Konstruktor soll die entsprechende Datei sofort einlesen. (Die zugehörige Datei muss bereits existieren.)
- `void` **`read`**`()`
  zum (ggf. erneuten) Einlesen aller Zeilen der Datei in eine passende Datenstruktur (Puffer) im Speicher.
- `void` **`write`**`()`
  zum Schreiben der gepufferten Inhalte zurück in die Datei.
- `int` **`availableLines`**`()`
  liefert die Anzahl der Zeilen im Puffer.
- `String[]` **`getLines`**`()`
  liefert alle Zeilen des Puffers als String-Array.
- `String` **`getLine`**`(int i)`
  liefert Zeile *`i`* des Puffers als String (Zählung beginnend bei Zeile 1). Die Methode soll eine eigene Exception `LineNumberOutOfBoundsException` werfen, wenn die Zeilenummer kleiner als 1 oder größer als die Anzahl der Zeilen im Puffer ist.
- `void` **`setLine`**`(int i, String s)`
  Setzt Zeile *`i`* des Puffers auf den String *s* (Zählung beginnend bei Zeile 1). Die Methode soll ebenfalls eine eigene Exception `LineNumberOutOfBoundsException` werfen, wenn die Zeilenummer kleiner als 1 oder größer als die Anzahl der Zeilen im Puffer ist.
- `void` **`replaceAll`**`(String regexp, String ersatz)`
  zum Ersetzen aller Vorkommen von `regexp` (darf ein regulärer Ausdruck sein!) gegen `ersatz` (in allen Zeilen des Puffers).
- `void` **`close`**`()`
  zum Schließen der Textdatei und Freigabe der internen Ressourcen (Puffer, File-Objekt).

Testen Sie die Klasse aus einer zweiten Klasse `TextFileTest`!
Fangen Sie dabei `LineNumberOutOfBoundsException`s ab!

Hinweis: Beachten Sie den Packagenamen!

Beispielinhalt der Datei „`bla.txt`“:

```
Alle meine Entchen
schwimmen auf dem See,
Köpfchen in das Wasser,
Schwänzchen in die Höh.
Alle meine Täubchen
gurren auf dem Dach
fliegt eins in die Lüfte
fliegen alle nach.
Alle meine Hühner
scharren in dem Stroh,
finden sie ein Körnchen,
sind sie alle froh.
Alle meine Gänschen
watscheln durch den Grund,
suchen in dem Tümpel,
werden kugelrund.
```

Nach Aufruf von: `aTextFile.replaceAll("meine", "unsre")`

```
Alle unsre Entchen
schwimmen auf dem See,
Köpfchen in das Wasser,
Schwänzchen in die Höh.
Alle unsre Täubchen
gurren auf dem Dach
fliegt eins in die Lüfte
fliegen alle nach.
Alle unsre Hühner
scharren in dem Stroh,
finden sie ein Körnchen,
sind sie alle froh.
Alle unsre Gänschen
watscheln durch den Grund,
suchen in dem Tümpel,
werden kugelrund.
```

Bereich: Input/Output (3)

Primzahlen speichern und lesen*	Schwierigkeit: ★★★☆☆
Package: `de.dhbwka.java.exercise.io`	**Klasse:** `PrimesFile`

Aufgabenstellung:

Schreiben Sie per Programm `PrimesFile` alle Primzahlen kleiner 100.000 in eine Datei „primes.txt“. Erweitern Sie dazu die Aufgabe „Sieb von Eratostenes“ um eine Ausgabekomponente, welche diese Datei erzeugt.

Schreiben Sie anschließend eine zweite Klasse `PrimesTest`, welche die Frage, ob eine eingegebene Zahl (<100.000) eine Primzahl ist, anhand der in dieser Datei gespeicherten Liste beantwortet.
Alternativ können Sie auch eine Reihe zufällig erzeugter Zahlen auf ihre Primzahl-Eigenschaft hin überprüfen.

Hinweis:
*Die *-Aufgabe ist für alle, die schon Erfahrung im Programmieren haben und/oder schon früher fertig geworden sind, manchmal anspruchsvoller, manchmal für Fleißige, manchmal vielleicht sogar ungelöst (für alle, die berühmt werden wollen…)*