DHBW Karlsruhe, Vorlesung Programmieren, „IO“

Vorab: Stellen Sie Ihre Entwicklungsumgebung so ein, dass alle Dateien dieser Übungsaufgaben in ein von Ihnen neu erstelltes Verzeichnis geschrieben und von dort gelesen werden.

(Z.B. in NetBeans 7.0: File → Project-Properties → Run → Working Directory)

Aufgabe 1 „File“

Schreiben Sie eine Klasse, welche in diesem Verzeichnis ein neues Verzeichnis „myDir“ anlegt und in diesem drei leere Dateien („foo1“, „foo2“ und „foo3“) anlegt.
Geben Sie weiterhin den absoluten Pfad sowie den Inhalt (die Dateinamen) des Verzeichnisses „myDir“ aus.

Kontrollieren Sie (z.B. im Windows-Explorer), ob die Dateien und das Verzeichnis tatsächlich angelegt wurden! Löschen Sie diese manuell!

Erweitern Sie anschließend die Klasse so, dass die drei Dateien und das Verzeichnis automatisch wieder gelöscht werden.

Aufgabe 2 „Quersumme“

Erweitern Sie die Quersummenaufgabe vom Aufgabenblatt „Strings“ derart, dass jede berechnete Quersumme und die zugehörige Zahl in ein mit einem Editor lesbares File geschrieben werden.

Aufgabe 3 „Palindrom“

Erweitern Sie die Palindromaufgabe vom Aufgabenblatt „Strings“ derart, dass jedes Palindrom in ein File geschrieben wird. Am Ende des Programmlaufs sollen dann alle gespeicherten Palindrome aus dem File ausgelesen und auf die Konsole ausgegeben werden.

Aufgabe 4 „Teil einer Datei“

Erzeugen Sie eine Textdatei mit dem Texteditor. Schreiben Sie in jede Zeile eine ganze Zahl. Fügen Sie so mindestens 8 Zeilen mit Zahlen ein.
Geben Sie per Java-Programm die Inhalte der Zeilen 3-6 dieser Datei sowie anschließend die Summe dieser 4 Zahlen aus!

Aufgabe 5 „TextFile“

Schreiben Sie eine Klasse `TextFile`. Diese soll folgende Methoden haben:

- Konstruktor `TextFile(File f)`
  zum Erzeugen eines neuen `TextFile`-Objektes.
  Der Konstruktor soll die Datei sofort einlesen.
- Konstruktor `TextFile(String pathname)`
  zum Erzeugen eines neuen `Textdatei`-Objektes.
  Der Konstruktor soll die Datei sofort einlesen.
- `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 verfügbaren Zeilen.
- `String[] getLines()`
  liefert alle Zeilen der Datei als String-Array.
- `String getLine(int i)`
  liefert Zeile *i* der Datei 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* der Datei 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.

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

**-Aufgabe*

*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…)*

Schreiben Sie alle Primzahlen kleiner 100.000 in eine Datei „prim.dat“. Erweitern Sie dazu die Aufgabe „Sieb von Eratostenes“ um eine Ausgabekomponente, welche diese Datei erzeugt.
Schreiben Sie eine zweite Klasse `PrimTest`, welche die Frage, ob eine eingegebene Zahl (<100.000) eine Primzahl ist, anhand der in dieser Datei gespeicherten Liste beantwortet.