Lösungen zu den Pflichtaufgaben Input / Output

1. Dateien und Attribute [PU]

Siehe Dokument *Lösungen zu den Übungsaufgaben Input / Output*.

2. Verstehen von Zeichensätzen [PU]

Siehe Dokument *Lösungen zu den Übungsaufgaben Input / Output*.

3. Byte- vs. Zeichenorientierte Streams [PU]

Siehe Dokument *Lösungen zu den Übungsaufgaben Input / Output*.

4. Picture File Datasource [PA]

a. Studieren Sie abgegebenen generischen und abstrakten Klassen, sowie die Klasse `Picture`, die bereits komplett implementiert ist.

b. Überlegen Sie sich, wie die einzelnen Operationen (insert, update, delete, ...) umgesetzt werden können, wenn sie mit zeilenweisen Records in einer Textdatei arbeiten:

   - Wie kann bei einem Insert die nächste noch nicht verwendete `id` bestimmt werden? Bedenken Sie:
     - Es kann sein das von verschiedenen Stellen auf die Datei zugegriffen wird. Sie können sich also nicht auf eine statische Variable verlassen.
     - Es können und dürfen beim Löschen von Records Lücken bei den id's entstehen
     - Die Zeilen müssen nicht geordnet sein, d.h. es muss nicht sein, dass der Record mit der grössten id am Ende steht.
     - Die Anzahl Zeilen ist kein guter Indikator, da wie gesagt die ids nicht immer fortlaufend sein müssen (d.h. Lücken von gelöschten Records haben kann).

> Am Sichersten ist es, wenn die Datei bei jedem Einfügen Zeile für Zeile gelesen und dabei jeweils die grösste Id festgestellt wird. Die nächste Id kann dann um eins grösser gewählt werden. Lücken aufzufüllen kann gefährlich sein, da man später ggf. zwischen neu eingefügt und geändert unterscheiden will. Die Zeile für den neuen Eintrag kann dann einfach am Ende in die Datei geschrieben werden. Um sicher zu gehen, dass nur ein Prozess gleichzeitig Änderungen vornimmt, müsste die Datei natürlich gesperrt werden. (see https://www.baeldung.com/java-lock-files)
>
> Alternativ, könnte ein separater Index erstellt werden, in welchem die aktuell grösste Id gespeichert wird oder sogar ein Mapping zwischen Ids und Zeilennummern (beim Einfügen und Löschen müsste dieser jedes Mal aktualisiert werden). Da von mehreren Stellen zugegriffen werden kann, müsste der Index jedoch auch als Datei gespeichert und aktualisiert werden.

- Wie aktualisieren Sie eine einzelne Zeile bei einem Update?

> Bei einem Update arbeitet man am einfachsten mit zwei Dateien. Aus der Originaldatei liest man zeilenweise ein. Wenn die id nicht mit derjenigen des zu aktualisierenden Datensatzes übereinstimmt, schreibt man die Zeile einfach in eine zweite temporäre Datei. Falls die id übereinstimmt, schreibt man die Zeile für den neuen Datensatz, anstelle der alten Zeile. Am Ende ersetzt man die Originaldatei durch die Kopie, sofern eine Änderung stattgefunden hat, ansonsten löscht man die temporäre Datei. Mit dieser Methode stellt man sicher, dass immer nur eine Zeile in den Speicher geladen wird, egal wie gross die Datei ist.
>
> Da jedes Mal alle Zeilen kopiert werden, ist die Methode eventuell nicht die schnellste, aber braucht wenig Speicher und ist relativ sicher, da die Originaldatei erst ganz am Schluss überschrieben wird. Bei einem Abbruch bleibt im schlimmsten Fall eine temporäre Datei liegen, die einfach gelöscht werden kann.
>
> In-Place-Änderungen z.B. mittels `RandomAccessFile` sind schwierig, da bei jedem Update die Zeilenlänge ändern kann und somit der ganze restliche Inhalt der Datei nach hinten oder vorne verschoben werden müsste, was auch langsam und zudem fehleranfällig ist.

- Wie entfernen Sie eine ganze Zeile bei einem Delete?

> Delete funktioniert ähnlich wie Update, ausser dass bei einer gefundenen id die Zeile übersprungen (d.h. nicht geschrieben) wird.

c. Implementieren sie die Methoden der Klasse `FilePictureDatasource`

- Nutzen sie die vorhandenen Konstanten und Hilfsobjekte (z.B, Dateformat)
- Beachten Sie die JavaDoc-Beschreibung der Methoden. Die Signatur der Methoden soll nicht verändert werden.
- Testen Sie ihre Implementation mit Hilfe der Klasse `PictureImport`, in welcher Bildinformationen von der Konsole abgefragt, als Picture-Record gespeichert und wieder

ausgelesen werden.

- Stellen Sie sicher, dass die Unit-Tests `FilePictureDataSourceTest` erfolgreich ausgeführt werden.

> Siehe Musterlösung im PictureDB Modul
>
> Die schreibenden Methoden sind wie oben beschrieben implementiert indem sie mit zwei Dateien arbeiten. Die lesenden Methoden verwenden direkt aus die Originaldatei.
>
> Als alternative Implementierung haben wir auch `FilePictureLambdaDatasource` beigefügt, welche anstelle der Leseschleifen jeweils mit Functional-Streams arbeitet.

d. Ergänzen Sie die Klasse `FilePictureDatasource` mit Logger-Meldungen.

- Die Initialisierung der Logger erfolgt über die Klasse `LogConfiguration`. Analysieren Sie die Konfiguration.
  - Welche Konfigurationsdatei wird geladen?

> Wenn vorhanden wird Konfigurationsdatei `log.properties` im PictureDB-Modulverzeichnis verwendet. Sie können es überschreiben, indem sie das SystemProperty `java.util.logging.config.file` setzen (z.B. in build.gradle`). Falls nicht vorhanden, wird eine Minimalkonfiguration von `log.properties` aus dem Ressourcen-Ordner verwendet.

  - Welche Log-Handler werden erzeugt und welche Meldungen wo ausgegeben?

> Es wird ein ConsolenHandler und ein FileHandler erzeugt, die vom dem Package-Logger `ch.zhw.prog2.io.picturedb` verwendet werden. Der Log-Level der Handler ist nicht begrenzt. Jedoch werden vom Package-Logger nur INFO-Meldungen entgegengenommen.

  - Wie kann das Format der Log-Meldungen angepasst werden?

> Bei Verwendung des `SimpleFormatter` kann das Format der Log-Meldungen durch das Property `java.util.logging.SimpleFormatter.format` festgelegt werden. Details zu den Argumenten findet man unter https://docs.oracle.com/en/java/javase/17/docs/api/java.logging/java/util/logging/SimpleFormatter.html und zur Formatierung der Daten hier https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Formatter.html

  - Wie können Sie die Konfiguration für die folgenden Anforderungen anpassen?

> siehe unten

- Verwenden Sie verschiedene Level von Log-Meldungen (INFO, WARNING, SEVERE, FINE,...). Zum Beispiel:

- Statusmeldungen → INFO (Record saved)
- Fehlermeldungen → WARNING oder SEVERE (Failed to save record)
- Debugmeldungen → FINE, FINER (New id=..., File opened/closed/copied/deleted)

> Siehe log Meldungen in der Musterlösung im PictureDB Modul

◦ Passen Sie die Logger-Konfiguration an

- Auf der Konsole sollen Meldungen des Levels INFO und höher ausgegeben werden.
- In eine zusätzliche Log-Datei `picturedb.log` sollen alle Meldungen (inkl. FINE, FINER) zeilenweise ausgegeben werden.

> Es müssen die Log-Level in der Konfiguration entsprechend angepasst werden.
>
> Handler müssen auf den vorgegebenen Level begrenzt werden:
>
> - `java.util.logging.ConsoleHandler.level = INFO`
> - `java.util.logging.FileHandler.level = ALL` oder `FINER`
>
> Logger müssen geöffnet werden, ansonsten wird bereits hier gefiltert:
>
> - `ch.zhaw.prog2.io.picturedb.level = ALL`
> - ggf. den Default-Level `.level = ALL`