Praktikum Input / Output

Einleitung

Ziele dieses Praktikums sind:

- Sie üben den Umgang mit Dateien und Dateiattributen.
- Sie verstehen Zeichensätze und können Sie anwenden.
- Sie beherrschen den Unterschied zwischen Byte- und Character-orientierten Streams.
- Sie können Inhalte aus Dateien lesen und schreiben.
- Sie können das Java Logger Framework anwenden.

Voraussetzungen

- Vorlesung Input/Output 1 und 2

Tooling

- Installiertes JDK 17+
- Gradle 7.6+

Struktur

Ein Praktikum kann verschiedene Arten von Aufgaben enthalten, die wie folgt gekennzeichnet sind:

[TU] – Theoretische Übung

Dient der Repetition bzw. Vertiefung des Stoffes aus der Vorlesung und als Vorbereitung für die nachfolgenden Übungen.

[PU] – Praktische Übung

Übungsaufgaben zur praktischen Vertiefung von Teilaspekten des behandelten Themas.

[PA] – Pflichtaufgabe

Übergreifende Aufgabe zum Abschluss. Das Lösen dieser Aufgaben ist Pflicht. Sie muss bis zum definierten Zeitpunkt abgegeben werden, wird bewertet und ist Teil der Vornote.

Zeit und Bewertung

Für das Praktikum stehen die Wochen gemäss den Angaben in Moodle zur Verfügung.
Je nach Kenntnis- und Erfahrungsstufe benötigen Sie mehr oder weniger Zeit. Nutzen Sie die Gelegenheit den Stoff zu vertiefen, auszuprobieren, Fragen zu stellen und Lösungen zu diskutieren (Intensive-Track).
Falls Sie das Thema schon beherrschen, müssen Sie nur die Pflichtaufgaben lösen und bis zum angegebenen Zeitpunkt abgeben (Fast-Track).

Die Pflichtaufgaben werden mit 0 bis 2 Punkten bewertet (siehe *Leistungsnachweise* auf Moodle).

Sie können die Lösungen der Textaufgaben in diesem Praktikum direkt in den Java-Files als Kommentar hinzufügen - oder als Markdown-File neben den Java-Files.

Referenzen

- Praktikumsverzeichnis – Quellcode, Projektstruktur

1. Dateien und Attribute [PU]

Im Praktikumsverzeichnis finden Sie das Projekt `FileAttributes`. Hier soll das Lesen von Dateien und Dateiattributen geübt werden.

Der Klasse `DirList` kann beim Starten ein Dateipfad übergeben werden. Falls der übergebene Pfad eine Datei ist, sollen die Attribute der Datei in einer Zeile ausgegeben werden. Falls es sich um ein Verzeichnis handelt, sollen die Attribute aller Dateien dieses Verzeichnisses zeilenweise ausgegeben werden.

Beispiel:

```
> java ch.zhaw.prog2.io.DirList .
frw-h 2020-02-24 16:49:57      630 .editorconfig
drwx- 2020-02-24 16:49:57       96 gradle
frw-- 2020-04-23 06:53:39     6392 README.adoc
frwx- 2020-02-24 16:49:57     5764 gradlew
...
```

Im ersten Block haben die Spalten folgende Bedeutung:

- Typ (File: `f` oder Directory: `d`),
- Leserecht (Read: `r` oder `-` falls nicht)
- Schreibrecht (Write: `w` oder `-` falls nicht)
- Ausführrecht (Execute: `x` oder `-` falls nicht)
- Versteckte Datei (Hidden: `h` oder `-` falls nicht)

Die nachfolgenden Blöcke enthalten:

- Datum und Uhrzeit der letzten Änderung
- Grösse der Datei in Bytes
- Name der Datei

Falls die übergebene Datei nicht existiert, soll eine Fehlermeldung ausgegeben werden.

Was bedeuteten die Attribute Lesen (`r`), Schreiben (`w`) und Ausführen (`x`) bei einem Verzeichnis?

2. Verstehen von Zeichensätzen [PU]

In der Vorlesung haben Sie gelernt, dass Java Unicode zum Speichern von Zeichen (Character) verwendet. Nun ist Unicode aber nicht der einzige Zeichensatz und Java unterstützt durchaus Alternativen zum Lesen und Schreiben. Welche Zeichensätze auf einem System konkret verwendet werden hängt von der Konfiguration des Betriebssystems und der JVM ab.

Im Praktikumsverzeichnis finden Sie das Projekt `Charsets`.

a. Ergänzen Sie in der Klasse `UnderstandingCharsets` den Code, um alle von der JVM unterstützten Zeichensätze auf der Konsole (`System.out`), sowie den für Ihr System definierten Standardzeichensatz auszugeben.
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/charset/Charset.html

b. Ergänzen Sie die Klasse so, dass sie einzelne Zeichen (also Zeichen für Zeichen) im Standardzeichensatz von der Konsole einliest und in zwei Dateien schreibt einmal im Standardzeichensatz und einmal im Zeichensatz `US-ASCII`.
   - Die Eingabe des Zeichens `q` soll das Program ordentlich beenden.
   - Die Dateien sollen `CharSetEvaluation_Default.txt` und `CharSetEvaluation_ASCII.txt` genannt werden und werden entweder erzeugt oder, falls sie bereits existieren, geöffnet und der Inhalt überschrieben.
   - Testen Sie Ihr Program mit den folgenden Zeichen: a B c d € f ü _ q
   - Öffnen Sie die Textdateien nach Ausführung des Programs mit einem Texteditor und erklären

Sie das Ergebnis.
- Öffnen Sie die Dateien anschliessend mit einem HEX-Viewer/Editor und vergleichen Sie.

Mit einem HEX-Viewer/Editor können die Bytes einer beliebigen Datei als Folge von Hexadezimalzahlen dargestellt und editiert werden. In der Regel könnend die Bytefolgen auch in Binär, Oktal oder als Zeichenkodierung angezeigt werden. Für die meisten IDE gibt es Hex-Viewer/Editoren als Plugins (z.B. BinEd). Alternativ können Sie diese auch unabhängig installieren (https://en.wikipedia.org/wiki/Comparison_of_hex_editors)

3. Byte vs. Zeichen-orientierte Streams [PU]

Im Unterricht haben Sie zwei Typen von IO-Streams kennengelernt; Byte- und Zeichenorientierte-Streams. In dieser Übung soll deren Verwendung geübt und analysiert werden was passiert, wenn der falsche Typ verwendet wird.

Im Praktikumsverzeichnis finden Sie das Projekt `ByteCharStream`, welches unter anderem das Verzeichnis `files` mit den Dateien `rmz450.jpg` und `rmz450-spec.txt` enthält. Ergänzen Sie die Klasse `FileCopy` mit folgender Funktionalität.

a. Verzeichnis-Struktur verifizieren: Methode `verifySourceDir()`
   - Das Quell-Verzeichnis soll auf Korrektheit überprüft werden.
   - Korrekt bedeutet, dass das Verzeichnis existiert und ausser zwei Dateien mit den Namen `rmz450.jpg` und `rmz450-spec.txt` nichts weiter enthält.
   - Im Fehlerfall werden Exceptions geworfen.
b. Dateien kopieren: Methode `copyFiles()`
   - Jede Datei im Quell-Verzeichnis soll zweimal kopiert werden, einmal zeichen- und einmal byte-orientiert.
   - Dazu soll die jeweilige Datei geöffnet und Element für Element (d.h. byte- bzw. charakterweise) von der Originaldatei gelesen und in die Zieldatei geschrieben werden.
   - Die Kopien sollen so benannt werden, dass aus dem Dateinamen hervorgeht, mit welcher Methode sie erstellt wurde.
c. Öffnen Sie die Kopien anschliessend mit einem entsprechenden Programm und erklären Sie die entstandenen Effekte.
d. Öffnen Sie die Kopien anschliessend mit einem HEX-Viewer/Editor und erklären Sie die Gründe für die Effekte.

4. Picture File Datasource [PA]

In Programmen will man oft die Anwendungslogik von der Datenhaltung (Persistenzschicht) abstrahieren, in dem ein Technologie-unabhängiges Interface zum Schreiben und Lesen der Daten verwendet wird. Dies ermöglicht den Wechsel zwischen verschiedenen Speichertechnologien (Datenbank, Dateien, Netzwerkserver, ...), ohne dass die Anwendungslogik angepasst werden muss.

In der Übung `PictureDB` verwenden wir ein Interface `PictureDatasource` zum Speichern und Lesen von Bildinformationen (Klasse `Picture`). `PictureDatasource` enthält Methoden, um auf eine Datenquelle zuzugreifen, welche Informationen zu Bildern speichert.

Vereinfacht sieht das Interface wie folgt aus:

```
public interface PictureDatasource {
    // inherited from GenericDatasource<T>
    public void insert(Picture picture);
    public void update(Picture picture) throws RecordNotFoundException;
    public void delete(Picture picture) throws RecordNotFoundException;
    public int count();
    public Picture findById(int id);
    public Collection<Picture> findAll();

    // extended finder method for looking up picture records
    public Collection<Picture>findByPosition(float longitude, float latitude, float
deviation);
}
```

In Realität erweitert `PictureDataSource` das generische Interface `GenericDatasource`, welches die Methoden enthält, die für alle Datenobjekttypen gleich sind, und definiert eine zusätzliche Picture spezifischen finder-Methode `findByPosition`.

Anhand der Methoden ist nicht ersichtlich, wie diese Informationen gespeichert werden. Es können somit unterschiedliche Implementationen für unterschiedliche Datenquellen implementiert werden (z.B. diverse Datenbanktypen SQL/No-SQL, Dateien, ...).

Ziel dieser Aufgabe ist es die Klasse `FilePictureDatasource` umzusetzen, welche Datensätze des Typs `Picture` in einer Datei verwaltet.

In der Datendatei (`db/picture-data.csv`) sollen die Daten der `Picture`-Objekte im *Character Separated Value* Format (CSV) gespeichert werden.

Das heisst jeder Datensatz wird in einer Zeile gespeichert. Die Felder werden mit einem Trennzeichen (`DELIMITER`), in unserem Fall der Strichpunkt (`;`) getrennt. Die Reihenfolge der Felder wird durch den bestehenden Inhalt der Datei vorgegeben.

```
id;date;longitude;latitude;title;url
```

Damit die Datensätze eindeutig identifiziert werden können, muss jeder Eintrag eine eindeutige Identifikation (`id`) besitzen, die sich, sobald gespeichert, nicht mehr ändern darf. Die `id` wird beim ersten Speichern in die Datasource von dieser bestimmt und im Datenobjekt gesetzt. Da jedes Datenobjekt diese Anforderung hat, wurde dies in der abstrakten Klasse `Record` implementiert, von welcher `Picture` abgeleitet ist.

`Record` hat nichts mit Java-Records zu tun. Es ist eine normale abstrakte Klasse, ist nicht final, d.h. kann / soll erweitert werden, und die Klassen benötigen einen Default-Konstruktor.

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).
- Wie aktualisieren Sie eine einzelne Zeile bei einem Update?
- Wie entfernen Sie eine ganze Zeile bei einem Delete?

Die Lösung muss mit einer minimalen, deterministischen Menge Speicher auskommen, d.h. Sie können nicht einfach die ganze Datei in den Speicher laden, da die Datei sehr gross sein könnte.

Da sie nicht gleichzeitig in der gleichen Datei lesen und schreiben können, hilft es gegebenenfalls mit zwei Dateien zu arbeiten (lesen → schreiben).

Die Klasse `java.nio.file.Files` bietet statische Hilfsmethoden zum Erstellen temporärer Dateien.

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.

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?
     - Welche Log-Handler werden erzeugt und welche Meldungen wo ausgegeben?
     - Wie kann das Format der Log-Meldungen angepasst werden?
     - Wie können Sie die Konfiguration für die folgenden Anforderungen anpassen?
   - 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)
   - 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.

Abschluss

Stellen Sie sicher, dass die Pflichtaufgaben mittels `gradle run` gestartet werden können, die Tests mit `gradle test` erfolgreich laufen und pushen Sie die Lösung vor der Deadline in Ihr Abgaberepository.