Aufgabe 1: Verlässlichkeit und I/O

Hinweis: Beachten Sie für die Abgabe auf jeden Fall die Anweisungen unter 1.6, sowie die Richtlinien für Code-Formatierung und Kommentare auf dem Hinweisblatt!

Die folgende Aufgabe soll die Grundlagen der Verlässlichkeit von Programmen, sowie der Ein- und Ausgabe noch einmal vertiefen. Dazu soll zunächst ein Key-Value-Store implementiert werden, der Schlüssel-Werte-Paare (engl. Key-Value-Pairs) speichern und verwalten kann. Daraufhin soll dieser mithilfe verschiedener Techniken verlässlicher gemacht werden. Dazu werden zum einen *Exceptions* hinzugefügt, mit deren Hilfe Ausnahmezustände erkannt und behoben werden können. Weiterhin sollen JUnit-Tests implementiert werden, welche die korrekte Funktionalität überprüfen. Zuletzt wird noch ein Checkpointing-System hinzugefügt, welches, mithilfe von Ein- und Ausgabe, den aktuellen Zustand des Key-Value-Stores auf der Festplatte sichern kann.

1.1 Key-Value-Store (6 Punkte)

In dieser Teilaufgabe sollen zunächst die Grundfunktionalitäten des Key-Value-Stores in der Klasse `KeyValueStore` implementiert werden. Der Key-Value-Store speichert dabei Key-Value-Pairs als Objekte der Klasse `KeyValuePair`, welche sowohl Key als auch Value als String-Objekte beinhalten. Gespeichert werden diese Objekte innerhalb eines Arrays, dessen Größe dem Konstruktor des Key-Value-Stores übergeben werden soll. Für den Zugriff auf die Key-Value-Pairs soll der Key-Value-Store vier Methoden bereitstellen:

- `newKVP(String key, String value)`: Legt ein neues Key-Value-Pair Objekt an und speichert dieses ab
- `getKVP(String key)`: Gibt den Value zu dem übergebenen `key` zurück
- `updateKVP(String key, String newValue)`: Setzt den Value zu dem bestehenden `key` auf `newValue`
- `deleteKVP(String key)`: Löscht das Key-Value-Pair des übergebenen Keys

Zusätzlich sollen diese Methoden bei ungültigen Parametern eine `IllegalArgumentException` geworfen werden. Diese soll zudem eine aussagekräftige Beschreibung des fehlerhaften Parameters beinhalten. Konkret ist ein Parameter fehlerhaft, wenn

- `null` für einen der Parameter übergeben wurde
- ein Key-Value-Pair mit dem gegebenen Key bei `newKVP` bereits existiert
- kein Key-Value-Pair für den gegebenen Key exisitert bei `get`/`update`/`delete`

1.2 Konsoleneingabe (4 Punkte)

In dieser Teilaufgabe soll nun ein kleines Programm implementiert werden, welches die Funktionalität des Key-Value-Stores nutzt, und eventuell auftretende Exceptions behandelt. Erweitern Sie dazu die Klasse `KVSMain` um eine `main`-Methode, welche mithilfe der `Scanner`-Klasse aus der Java-Standardbibliothek Nutzereingaben von der Konsole entgegennimmt, auswertet, und entsprechende Methoden auf dem Key-Value-Store aufruft. Das Programm soll so lange weitere Befehle über die Konsole annehmen bis es vom Benutzer beendet wird. Dabei soll für jede der Grundmethoden des Key-Value-Stores ein Befehl auf der Kommandozeile existieren, bswp. `new mykey myvalue`, oder `get mykey`. Zusätzlich soll es einen `exit` Befehl geben, der das Programm beendet.

Nach jeder erfolgreichen Ausführung eines Befehls soll dann „ok“, bzw. im Falle eines `get`-Befehls soll stattdessen der Wert des zurückgegebenen Values ausgegeben werden. Falls eine Exception ausgelöst wird, soll das Programm die Nachricht ausgeben, welche in der Exception enthalten ist, und danach weitere Befehle entgegennehmen.

1.3 Checkpointing (6 Punkte)

In der folgenden Teilaufgabe soll der Key-Value-Store nun gegen eventuelle Abstürze abgesichert werden. Dazu soll ein Checkpointing implementiert werden, welches den Zustand des Programmes (sprich alle Keys und Values) in regelmäßigen Abständen in einer Datei auf der Festplatte speichert. Wird das Programm dann neu gestartet, kann es den letzten Zustand von dieser Datei auslesen, und die enthaltenen Key-Value-Pairs wiederherstellen.

Dazu sollen zwei Klassen `KVPInputStream` und `KVPOutputStream` implementiert werden, welche direkt von `InputStream` bzw. `OutputStream` erben. Die Klasse `KVPOutputStream` soll die Methode `writeKVP` enthalten, welche ein Key-Value-Pair übergeben bekommt, und dieses mithilfe eines `FileOutputStreams` in eine Datei schreibt. Der `KVPInputStream` soll hingegen eine entsprechende `readKVP` Methode implementieren, welche das nächste in der Datei enthaltene Key-Value-Pair ausliest und zurückgibt. Um die korrekte Funktionalität zu gewährleisten sollte die Datei also nur ein Mal bei der Erstellung oder Wiederherstellung von einem Checkpoint, aber nicht bei jedem Aufruf von `write` oder `read` geöffnet werden. **Für das erwartete Format der Datei**

halten Sie sich bitte an die Vorgaben aus der ersten großen Übung! Zudem ist die Nutzung der `ObjectStream`-Klassen explizit untersagt. Für den Fall, dass ein Fehler beim öffnen oder auslesen der Datei auftritt, oder das Format der Datei nicht den Vorgaben entspricht, sollen die Methoden zudem entsprechende `IOExceptions` werfen.

Weiterhin soll die `KeyValueStore`-Klasse angepasst werden, sodass diese nach jeweils 5 Operationen, welche den Zustand verändern (`new`/`update`/`delete`) alle Key-Value-Pairs in die Datei schreiben. Zuletzt soll bei der Erstellung eines Key-Value-Stores versucht werden die Checkpoint-Datei einzulesen und den Zustand wiederherzustellen. Ist dies nicht möglich, soll stattdessen wie bisher ein leerer Key-Value-Store erzeugt werden.

1.4 JUnit Tests (4 Punkte)

In der letzten Teilaufgabe sollen nun JUnit-Tests geschrieben werden, welche die vier Grundmethoden des Key-Value-Stores testen. Implementieren sie hierfür in der Klasse `KVSTest` für jede Grundmethode einen Test, welcher je einen Normal- und einen Fehlerfall überprüft.

1.5 Formattierung (1 Punkte)

Für die Umsetzung von der obigen Teilaspekte erhalten Sie entsprechend Punkte. Haben Sie mindestens drei Teilpunkte erhalten, können Sie einen weiteren Punkt für Einhaltung der Programmierrichtlinien erhalten. Diese sind:

- Formatierung des Programms gemäß der Richtlinien.
- Javadoc Kommentare für jede Klasse, jedes Interface und jede public Methode/Funktion.
- Keine überflüssigen Dateien (hier primär `.class` und `.jar` Datein, aber auch Sonstiges, das nicht unter Quelltext oder Dokumentation fällt) im Git.

1.6 Abgabe

Committen und pushen Sie alle modifizierten Dateien in Ihr Git Repository. Teil Ihrer Abgabe soll auch eine ausgefüllte „Teilnehmer.txt“-Datei sein, die ebenfalls in Ihr Git gepusht werden soll. Diese soll ihren Namen, Matrikelnummer und Studiengang (bspw: Bachelor Informatik) enthalten. Ohne diese kann die erfolgreiche Bearbeitung der Hausaufgaben nicht an das zuständige Prüfungsamt weitergeleitet werden!

Abgabe bis 15. Mai, 23:59 Uhr.

Ihre Lösungen sollen bis zur Deadline im GitLab des IAS hochgeladen sein.
Insgesamt zu erzielende Punktzahl: 21 Punkte
Zu erzielende Minimalpunktzahl: 7 Punkte