2025/11/21 11:15 1/4 LU02a - Konsoleneingabe

# LU02a - Funktionale Implementationen 1

### **Main-Methode**

Die Main-Methode ist der Ausgangspunkt einer Java-Applikation.

```
public static void main(String[] args) {
    ...
}
```

Das String-Array args, welches der Methode übergeben wird, entsprechen den Argumenten, wenn das kompilierte Programm aufgerufen wird.

Erstellen Sie ein Fork-Repository mit allen Branches von <a href="https://github.com/bzz-templates-java/ffit-lu02-library-app">https://github.com/bzz-templates-java/ffit-lu02-library-app</a> Klonen Sie anschliessend ihr eigenes Repository, um die Basisstruktur für die LibraryApp zu erhalten, welche Sie in den kommenden Kapiteln implementieren werden.

#### Scanner

Bei vielen Backend-Applikationen erfolgt die Benutzerinteraktionen via API-Aufrufen oder anderen Schnittstellen. Für den Anfang können wir die Klasse Scanner nutzen. Diese erlaubt es, Benutzereingaben in der Konsole zu lesen.

Nutzen Sie die Klasse Scanner, um einen String einzulesen.

- https://javabeginners.de/Ein- und Ausgabe/Scanner.php
- https://www.w3schools.com/java/java user input.asp

Diesen String verwenden wir als Befehl, um verschiedene Implementierte Funktionen auszuführen.

**Anforderung 1**: Die Eingabe quit soll das Programm beenden, die Eingabe help soll die Liste aller implementierten Befehle auflisten und bei einer anderen Eingabe soll eine Nachricht ausgegeben werden, dass die Eingabe nicht als Befehl erkannt wurde. Nach dem Ausführen eines Befehls (ausser quit) soll der nächste eingegeben werden können usw.

Überlegen Sie sich vor der Implementation, wie der Code so gestaltet werden kann, dass zukünftige Befehle einfach hinzugefügt werden können. Vielleicht haben Sie bereits eine Idee, wie man die Befehle erfassen könnte, so dass die Liste der Befehle (help) nicht separat erweitern muss. Sie können aber solche Verbesserungen auch später noch erledigen.

Machen Sie einen Cherry-Pick des Commits d5cca5d, um die Tests für diesen Implementationsschritt zu erhalten. Führen Sie die Tests aus und korrigieren Sie ihre Implementation falls nötig.

Sollte die Fehlermeldung bad revision auftauchen, checken Sie den anderen Branch aus, überprüfen Sie, ob der Commit in den Logs vorhanden ist und wechseln Sie wieder zurück auf den Main-Branch.

```
git checkout tests
git log
git checkout main
<code>
==== Objekte ====
**Anforderung 2**: Eine Bibliotheksapplikation soll die vorhanden Bücher mit
dem Befehl ''listBooks'' auflisten können.
Dazu braucht es erstmal eine Klasse ''Book''. Erstellen Sie die Klasse und
die folgenden beiden Objekte als Konstanten. Beim Befehl ''listBooks''
sollen diese beiden Bücher je ein Buch(mindestens der Titel) pro Zeile
ausgegeben werden.
^id ^isbn ^title ^author ^year |
|1|978-3-8362-9544-4|Java ist auch eine Insel|Christian Ullenboom|2023|
|2|978-3-658-43573-8|Grundkurs Java|Dietmar Abts|2024|
Machen Sie einen Cherry-Pick des Commits ''1766cfc'', um die Tests für
diesen Implementationsschritt zu erhalten. Führen Sie die Tests aus und
korrigieren Sie ihre Implementation falls nötig.
==== Persistierung ====
Anstatt die Bücher nur hardcodiert in der Applikation zu haben, soll die
PostgreSQL-Datenbank verwendet werden.
Loggen Sie sich mit folgendem Befehl bei der Datenbank an und setzen Sie
nachfolgende SQL-Statements ab, um einige Beispieleinträge zu erfassen.
<code bash>
psql -U localuser -d localdb
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    isbn VARCHAR(20) NOT NULL,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    publication year INT
);
INSERT INTO books (isbn, title, author, publication year)
```

Mit \quit können Sie sich übrigens wieder von PostgreSQL abmelden.

**VALUES** 

https://wiki.bzz.ch/ Printed on 2025/11/21 11:15

('978-0596009205', 'Head First Java', 'Kathy Sierra, Bert Bates', 2005);

('978-0134685991', 'Effective Java', 'Joshua Bloch', 2018),

2025/11/21 11:15 3/4 LU02a - Konsoleneingabe

#### \quit

Für den Zugang zu einer Datenbank verwenden wir JDBC (Java Data-Base Connectivity).

https://www.baeldung.com/java-jdbc

Anstatt der Maven-Dependency muss die Dependency in build.gradle eingetragen werden. Sie erhalten automatisch die nachfolgende Liste, wenn Sie den Commits 23e74a3 cherry-picken.

```
dependencies {
    // dependencies for the application
    implementation 'org.postgresql:postgresql:42.7.3'

    // dependencies for testing
    testImplementation 'org.junit.jupiter:junit-jupiter:5.10.2'
}
```

Implementieren Sie eine Methode, die eine Datenbankverbindung aufbaut, folgende Abfrage macht und für jeden zurückgelieferten Eintrag ein Book-Objekt erstellt. Die Methode soll dann eine Liste (java.util.List) der Buch-Objekte zurückgeben.

```
SELECT id, isbn, title, author, publication_year FROM books
```

Führen Sie die Tests aus und korrigieren Sie ihre Implementation falls nötig.

# Konfigurationsvariabeln

URLs, Benutzernamen und Passwörter sollten nicht hartcodiert werden, da sonst jeder mit Read-Rechten die Zugangsdaten lesen und somit auf die Datenbank zugreifen kann.

Es gibt verschiedene Arten, um solche Variabeln einzulesen, wie zum Beispiel: Command-line Argumente, System Properties, eine .env-Datei, usw.

Für native Java eignet sich java.util.Properties

- https://www.baeldung.com/java-properties
- 1. Erstellen Sie die Datei config.properties anhand von config.properties.template.
- 2. Initialisieren Sie ein Properties-Objekt:

```
new Properties()
```

3. Laden Sie die Datei aus dem Root-Verzeichnis:

```
new FileInputStream("config.properties")
```

4. Holen Sie die Werte der Konfigurationen von DB\_URL, DB\_USER und DB\_PASSWORD aus den Properties.

config.properties selbst wird nicht im Git-Repository eingecheckt, sondern existiert nur lokal. So ist sichergestellt, dass sensible Angaben nicht "geleakt" werden.

Führen Sie die Tests erneut aus und korrigieren Sie ihre Implementation falls nötig.

Für die Pipeline wird eine separate Testdatenbank benötigt. Studieren Sie die Ergänzungen der Pipeline und die beiden SQL-Dateien, um den Einsatz der Postgres-Service zu verstehen.

#### 1/0

Stellen Sie sich vor, dass die Betreiber einer Bibliothek bisher eine Excel-Datei für die Verwaltung der Bücher genutzt haben [].

Anstatt alle darin enthaltenen Bücher abzutippen und manuell in die Datenbank einzutragen, möchten sie eine Import-Funktionalität. Da die Buchtitel teilweise Kommas und Semikolons enthalten, verwenden wir \t als Trennzeichen für den Export. Dieser ist zudem in UTF-8, was auch die Standardcodierung in Git ist.

**Anforderung 3**: Mit dem Befehl importBooks <FILE\_PATH> soll die angegebene TSV-Datei eingelesen werden und in die Datenbank gespeichert werden. <FILE\_PATH> steht dabei natürlich für einen Pfad und Dateinamen.

- Erweitern Sie die Befehlslogik, so dass der Befehl importBooks mit dem Pfad als Argument erkannt wird.
- Schreiben Sie die Logik, um die Datei ohne zusätzliche Dependencies einzulesen und eine List<Book> daraus zu erstellen. Beispielcode für einen solchen Reader finden Sie u.a. auf https://www.baeldung.com/java-csv-file-array
- 3. Speichern Sie die resultierende Liste in die Datenbank. Einträge mit derselben id sollen überschrieben werden, damit man auch Korrekturen mit dieser Funktion einlesen kann. https://www.baeldung.com/java-jdbc
- 4. Importieren Sie die Datei books.tsv im Ordner data

importBooks data/books.tsv

5. Überprüfen Sie die Anforderung mit den Tests im Commit 670f42e

## Weitere Anpassungen (optional)

**Change request 2.1**: Damit listBooks nicht immer alle Bücher ausgibt, soll optional ein Limit mitgegeben werden können. listBooks 10 soll also maximal die ersten 10 Bücher zurückliefern.

From:

https://wiki.bzz.ch/ - BZZ - Modulwiki

Permanent link:

https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu02/main?rev=1756193374

Last update: 2025/08/26 09:29



https://wiki.bzz.ch/ Printed on 2025/11/21 11:15