Name:	
Klasse:	
Datum:	

Schuljahr: 2022/23

notwendige Vorkenntnisse

- OOP-Grundlagen
- Lambdaausdrücke
- MariaDB ☞
- SQL ☞
- Maven ☞
- JDBC ☞

1 JDBI – Java Database idiomatic access

JDBI baut auf JDBC auf und verbessert dabei deutlich die Schnittstelle zur Datenbank und bietet eine modulare Erweiterung durch Plugins an. Es wird dabei kein komplett objektrelationales Mapping zur Verfügung gestellt, sondern eine einfache Möglichkeit, das Mapping zwischen Relationen und Objekten so zu konstruieren, wie es für Ihre Anwendung angemessen ist.

1.1 Eclipse-Projekt

Importieren Sie das Eclipse-Projekt `eclipse_db_jdbi_01.zip`.
Voraussetzung: OpenJDK 17 (☞Installationsanleitung) und ein aktuelles Eclipse EE (☞Installationsanleitung), z. B. 2022-09 oder neuer.

1.2 Beispieldatenbank

Als Datenbank wird hier `bsp` verwendet. Mit nachfolgenden SQL-Anweisungen wird diese erzeugt.

```
# create db bsp, user bsp - kein PW
USE mysql;

DROP DATABASE IF EXISTS bsp;
CREATE DATABASE bsp CHARACTER SET utf8;

DROP USER IF EXISTS 'bsp'@'localhost';
CREATE USER 'bsp'@'localhost' IDENTIFIED BY '';

GRANT ALL PRIVILEGES ON bsp.* TO 'bsp'@'localhost';
FLUSH PRIVILEGES;
```

Aufgabe JDBI-01-1

Richten Sie die Datenbank ein.

Notiz:

1.3 Verbindungsaufbau und Erstellung der Tabelle

Für JDBI sind einige Bibliotheken notwendig:

- lombok für getter/setter
- MariaDB JDBC-Treiber
- JDBI und weitere Plugins
- Logging-Framework (verwendet JDBI)

```
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <version>1.18.22</version>
   <scope>provided</scope>
</dependency>

<dependency>
   <groupId>org.mariadb.jdbc</groupId>
   <artifactId>mariadb-java-client</artifactId>
   <version>3.0.7</version>
</dependency>

<dependency>
   <groupId>org.jdbi</groupId>
   <artifactId>jdbi3-core</artifactId>
   <version>3.34.0</version>
</dependency>

<dependency>
   <groupId>org.jdbi</groupId>
   <artifactId>jdbi3-sqlobject</artifactId>
   <version>3.34.0</version>
</dependency>

<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-api</artifactId>
   <version>1.7.36</version>
</dependency>
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-simple</artifactId>
   <version>1.7.36</version>
</dependency>
```

Der Code

```
10  final Jdbi jdbi = Jdbi.create("jdbc:mariadb://localhost:3306/bsp", "bsp", "");
11  System.out.println("Connect bsp ...");
12
13  final Handle handle = jdbi.open();
14
15  handle.execute("DROP TABLE IF EXISTS mitarbeiter");
16
17  handle.execute("""
18        CREATE OR REPLACE TABLE mitarbeiter (
19          id      INT         PRIMARY KEY AUTO_INCREMENT,
20          name    VARCHAR(50),
21          vorname VARCHAR(50),
22          gebdat  DATE,
23          strasse VARCHAR(50)
24        ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
25                          """);
26  handle.execute("INSERT INTO mitarbeiter (name,vorname) VALUES ('Kobold', 'Pumukel')");
27  handle.execute("INSERT INTO mitarbeiter (name,vorname) VALUES ('Müller', 'Max')");
28  handle.execute("INSERT INTO mitarbeiter (name,vorname) VALUES ('Meier', 'Uschi')");
29  handle.execute("INSERT INTO mitarbeiter (name,vorname) VALUES ('Schlau', 'Susi')");
30
31  System.out.println("Tabelle 'mitarbeiter' erstellt.");
```

Wie bei JDBC muss hier erstmal eine Verbindung zur Datenbank aufgebaut werden. Dazu wird über die Methode `create()` die URI, der User und sein PW an JDBC weitergeben und ein zentrales JDBI-Objekt erzeugt.

Ein Handler für das weitere Arbeiten wird mit `open()` erzeugt.

Anschließend wird eine evtl. vorhandene Tabelle gelöscht und dann neu erstellt. Zusätzlich werden ein paar Einträge in der Tabelle hinzugefügt.

1.4 Datensätze ausgeben

Datensätze werden über `createQuery()` abgefragt.

```
13      final Jdbi jdbi = Jdbi.create("jdbc:mariadb://localhost:3306/bsp", "bsp", "");
14      System.out.println("Connect bsp ...");
15
16      final Handle handle = jdbi.open();
17
18      final List<Map<String, Object>> results = handle
19            .createQuery("SELECT * FROM mitarbeiter")
20            .mapToMap()
21            .list();
22
23      System.out.println("######################");
24      results.stream().forEach(e -> System.out.println(e));
```

Die Ausgabe sieht dabei wie folgt aus:

```
Connect bsp ...
######################
{id=1, name=Kobold, vorname=Pumukel, gebdat=null, strasse=null}
{id=2, name=Müller, vorname=Max, gebdat=null, strasse=null}
{id=3, name=Meier, vorname=Uschi, gebdat=null, strasse=null}
{id=4, name=Schlau, vorname=Susi, gebdat=null, strasse=null}
```

Der große Vorteil gegenüber JDBC ist hier, dass nicht aufwendig bei jeder Zeile der Wert der Spalte ermittelt werden muss, sondern einfach mit der Funktionalität von lambda-Ausdrücken gearbeitet werden kann.

1.5 Datensätze einfügen

Beim Einfügen von Datensätzen wird hier standardmäßig mit einen PreparedStatement gearbeitet.

```
14      final Jdbi jdbi = Jdbi.create("jdbc:mariadb://localhost:3306/bsp", "bsp", "");
15      System.out.println("Connect bsp ...");
16
17      final Handle handle = jdbi.open();
18
19      final int result = handle.execute("""
20            INSERT INTO mitarbeiter (name, vorname, gebdat, strasse)
21               VALUES (?, ?, ?, ?)
22            """, "Huber", "Peter", new Date(), "Teststr. 23");
23
24      System.out.println("Datensätze geändert: " + result);
25
26      final List<Map<String, Object>> results = handle.createQuery("SELECT * FROM mitarbeiter")
27            .mapToMap()
28            .list();
29
30      System.out.println("######################");
31      results.stream()
32            .forEach(e -> System.out.println(e));
```

Die Ausgabe sieht dabei wie folgt aus:

```
Connect bsp ...
1
######################
{id=1, name=Kobold, vorname=Pumukel, gebdat=null, strasse=null}
{id=2, name=Müller, vorname=Max, gebdat=null, strasse=null}
{id=3, name=Meier, vorname=Uschi, gebdat=null, strasse=null}
{id=4, name=Schlau, vorname=Susi, gebdat=null, strasse=null}
{id=5, name=Huber, vorname=Peter, gebdat=2022-10-20, strasse=Teststr. 23}
```


1.6 Datensätze ändern und löschen

Ein Update bzw. ein Delete verhält sich wie ein Insert.

```
13      final Jdbi jdbi = Jdbi.create("jdbc:mariadb://localhost:3306/bsp", "bsp", "");
14      System.out.println("Connect bsp ...");
15
16      final Handle handle = jdbi.open();
17
18      final int result = handle.execute("""
19            DELETE FROM mitarbeiter WHERE name='Huber'
20            """);
21
22      System.out.println("Datensätze geändert: " + result);
23
24      final List<Map<String, Object>> results = handle.createQuery("SELECT * FROM mitarbeiter")
25            .mapToMap()
26            .list();
27
28      System.out.println("######################");
29      results.stream()
30            .forEach(e -> System.out.println(e));
```

Die Ausgabe sieht dabei wie folgt aus:

```
Connect bsp ...
1
######################
{id=1, name=Kobold, vorname=Pumukel, gebdat=null, strasse=null}
{id=2, name=Müller, vorname=Max, gebdat=null, strasse=Burgweg 12}
{id=3, name=Meier, vorname=Uschi, gebdat=null, strasse=null}
{id=4, name=Schlau, vorname=Susi, gebdat=null, strasse=null}
{id=5, name=Huber, vorname=Peter, gebdat=2022-10-20, strasse=Teststr. 23}
```

Aufgabe JDBI-01-2

Probieren Sie die Beispiele selbst aus.

Notiz: