Wie kann in Java eine lokale SQLite-Datenbank genutzt werden?

Eine SQLite-Datenbank stellt eine lokale Datenbank dar, die ohne Server betrieben und über die weit verbreitete Datenbanksprache *SQL* gesteuert wird. Nach Einbindung des entsprechenden JDBC-Treibers kann sie auf einfache Weise aus Java heraus erzeugt und angesprochen werden.

Um das Beispiel erfolgreich ausführen zu können, muss zunächst der passende Datenbank-Treiber heruntergeladen und in den Classpath des Programms eingebunden werden.

Die Beispielklasse selbst ist als Singleton realisiert. Sie kann so zu einer Klasse zur zentralen Datenbanksteuerung ausgebaut werden, von der nur ein Objekt programmweit existieren sollte.

Gleich zu Beginn werden drei Instanzvariablen deklariert. Die erste ist mit dem Singleton-Objekt der Klasse initialisiert, die zweite ein `Connection`-Objekt, das später zum Verbindungsaufbau zur Datenbank genutzt wird und die dritte enthält den Pfad zur Datenbank-Datei. Er kann innerhalb der Rechte, die dem Programmnutzer zustehen, frei gewählt werden.
Es folgt ein statischer Block, in dem der Datenbanktreiber geladen wird, sowie der leere, hier beim Singleton `private` deklarierte Konstruktor.
In der `main`-Methode werden zum Ausführen des Programms nacheinander die beiden realisierten Methoden aufgerufen: `initDBConnection()` baut die Verbindung zur Datenbank auf und schließt sie bei Programmende, `handleDB()` erzeugt einige Einträge und fragt sie anschließend zu Demonstrationszwecken ab.

Der Reihe nach! Nach einer Sicherheitsabfrage, die geährleistet, dass eine bereits bestehende Verbindung nicht erneut aufgebaut wird, findet der Verbindungsaufbau durch die statische Methode `getConnecion()` der Klasse `DriverManager` statt. Ihr wird eine URL als Parameter übergeben, dessen letzter Teil der Pfad zur Datenbankdatei ist.
Wie oben erwähnt, kann der Pfad im Rahmen der Rechte des Users frei gewählt werden. Die Datei selbst wird automatisch neu gebildet, wenn sie nicht bereits existiert. Der gegen Fehler beim Verbindungsaufbau durch eine Exception abgesicherte Block wird durch eine Kontrollausgabe des `Connecion`-Objectes abgeschlossen.
In der nächsten Anweisungsfolge wird ein neuer Thread gebildet,

innerhalb dessen die bestehende Verbindung geschlossen wird. Er bleibt zunächst ungestartet, und wird als Parameter dem aktuellen `Runtime`-Objekt als *'Shutdown-Hook'* übergeben. Dies bedeutet, dass die Viruelle Maschine beim Herunterfahren diesen Thread startet und somit die Datenbankverbindung sicher schließt.

```
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class DBController {

    private static final DBController dbcontroller = new DBC
    private static Connection connection;
    private static final String DB_PATH = System.getPropert

    static {
        try {
            Class.forName("org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            System.err.println("Fehler beim Laden des JDBC-
Treibers");
            e.printStackTrace();
        }
    }

    private DBController(){
    }

    public static DBController getInstance(){
        return dbcontroller;
    }

    private void initDBConnection() {
        try {
            if (connection != null)
                return;
            System.out.println("Creating Connection to Datab
            connection = DriverManager.getConnection("jdbc:s
            if (!connection.isClosed())
                System.out.println("...Connection establishe
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        Runtime.getRuntime().addShutdownHook(new Thread() {
```

```
            public void run() {
                try {
                    if (!connection.isClosed() && connectio
                        connection.close();
                        if (connection.isClosed())
                            System.out.println("Connection
                    }
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        });
    }

    private void handleDB() {
        try {
            Statement stmt = connection.createStatement();
            stmt.executeUpdate("DROP TABLE IF EXISTS books;
            stmt.executeUpdate("CREATE TABLE books (author,
            stmt.execute("INSERT INTO books (author, title,
05-06', '1234', '5.67')");

            PreparedStatement ps = connection
                    .prepareStatement("INSERT INTO books VA

            ps.setString(1, "Willi Winzig");
            ps.setString(2, "Willi's Wille");
            ps.setDate(3, Date.valueOf("2011-05-16"));
            ps.setInt(4, 432);
            ps.setDouble(5, 32.95);
            ps.addBatch();

            ps.setString(1, "Anton Antonius");
            ps.setString(2, "Anton's Alarm");
            ps.setDate(3, Date.valueOf("2009-10-01"));
            ps.setInt(4, 123);
            ps.setDouble(5, 98.76);
            ps.addBatch();

            connection.setAutoCommit(false);
            ps.executeBatch();
            connection.setAutoCommit(true);

            ResultSet rs = stmt.executeQuery("SELECT * FROM
            while (rs.next()) {
                System.out.println("Autor = " + rs.getStrin
                System.out.println("Titel = " + rs.getStrin
                System.out.println("Erscheinungsdatum = "
                        + rs.getDate("publication"));
                System.out.println("Seiten = " + rs.getInt("
                System.out.println("Preis = " + rs.getDoubl
            }
            rs.close();
```

```
            connection.close();
        } catch (SQLException e) {
            System.err.println("Couldn't handle DB-
Query");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        DBController dbc = DBController.getInstance();
        dbc.initDBConnection();
        dbc.handleDB();
    }
}
```

Innerhalb der Methode `handleDB()` wird als erstes ein `Statement`-Objekt erzeugt. Auf ihm werden drei SQL-Anweisungen ausgeführt, die nacheinader zunächst eine eventuell existierende Tabelle *'books'* löscht, sie anschließend wieder neu erzeugt und schließlich in diese Tabelle einen einzelnen Datensatz einträgt.
Es fällt auf, dass hier zwei verschiedene Methoden verwandt werden:

- `executeUpdate()` führt Anfragen aus, die, wie bspw. *INSERT, UPDATE* oder *DELETE*, nichts zurückgeben.
- `execute()` ist mehrfach überladen und kann mehrere Ergebnisse liefern. Die Methode gibt einen boolschen Wert zurück der `true` ist, wenn ein `ResultSet` erzeugt wurde und `false`, falls entweder gar keine Rückgabe oder die Anzahl der geänderten Datensätze geliefert wurde.

Die folgenden Zeilen zeigen die Ausführung eines *prepared Statements*, einer Abfrageform, die besonders gut geeignet ist, um mehrere Abfragen nacheinander auszuführen.
Das `PreparedStatement`-Objekt wird direkt auf dem Verbindungs-Objekt erzeugt. Ihm wird bei der Bildung direkt der Abfrage-String übergeben. Hier handelt es sich um ein konventionelles *INSERT*-Statement, das jedoch statt der Werte Fragezeichen als Platzhalter enthält. Sie werden bei der Ausführung des Statements durch die konkreten Werte ersetzt. Diese Übergabe erfolgt durch Setter-Methoden, die für die verschiedenen Datentypen zur Verfügung stehen. Der erste Parameter dieser Methoden bezeichnet die Spalte, der zweite den Wert des Eintrags.

Die Methode `addBatch()` fügt den erzeugten Auftrag einem Stapel an Querys hinzu, der schließlich als Gesamtheit durch `executeBatch()` ausgeführt wird. Vorher wird der Auto-Commit-Modus auf `false` gesetzt, um diesen Anweisungs-Stapel als geschlossene Einheit auszuführen.

Die Abfrage der eingetragenen Datensätze erfolgt mittels einer weiteren execute...-Methode, `executeQuery()`. Sie liefert ein `ResultSet`-Objekt, das in einer Schleife ausgelesen wird und die

Daten mit Hilfe diverser Getter-Methoden liefert. Die Datenbank-Spaltennamen werden in Form von Strings als Parameter übergeben.

Den Abschluss bilden die Schließung des `ResultSets` und der Datenbank-Verbindung.