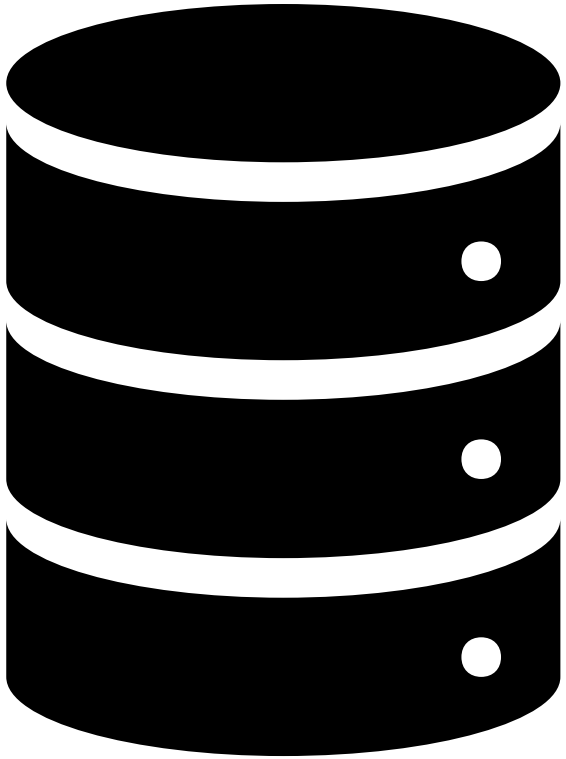


JDBC

März 2025

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Aggregate • JDBC • SQL-Injection	• Implementierungs-weisen • BestPractises • MVC-Architektur • ER-Modelle	• Wiederholung	• Projekt	• Projekt

SMART INDUSTRY CAMPUS

Plan für heute

- Projekt auf Git pushen
- Projekt auf Git als Module hinterlegen
- Wiederholen der erlernten Konzepte

SMART INDUSTRY CAMPUS

Schritt 1

SMART INDUSTRY CAMPUS

Bisheriger Stand

- SQL-Queries werden genutzt, um eine Anfrage in der Datenbank zu machen
- bisher: Anfragen über die MySQL Workbench
- im realen Arbeitsleben werden Anfragen nicht mehr über eine Workbench getätigt, sondern direkt in einer Anwendung eingebaut
- Wie kann man SQL-Queries in eine Java-Anwendung einbinden?
  - Über JDBC!

JDBC

- ist eine API (Schnittstelle), die es Java-Anwendungen ermöglicht, mit relationalen Datenbanken zu kommunizieren
- = JDBC dient als Kommunikationsschnittstelle zwischen Anwendung und Datenbank

DriverManager

- Stellt eine Verbindung zur Datenbank her
- wichtige Methode:

  getConnection(String url, String user, String password)

  - Erzeugt eine Verbindung (= Connection) zur Datenbank

  - URL zur Datenbank — jdbc:mysql://localhost:3306/myDatabaseName
  - User der Datenbank: — root
  - Passwort der Datenbank — übergebenes Passwort

Connection

Warum try/catch?

Beim Aufbau einer Verbindung kann etwas schief gehen -> Behandlung mittels try/catch

- repräsentiert eine Verbindung zur Datenbank
- wird in Kombination mit dem DriverManager genutzt:
  - Der DriverManager holt sich die Verbindung anhand der URL, USER und PASSWORD
- eine Connection sollte am besten einmal erstellt und dann wiederverwendet werden!
- Beispiel:

```
private static final String URL = "jdbc:mysql://localhost:3306/hospital";
private static final String USER = "root";
private static final String PASSWORD = "password";

public static Connection getConnection() {
    try {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    } catch (SQLException ex) {
        System.out.println("Verbindung konnte nicht hergestellt werden");
    }
    return null;
}
```

SMART INDUSTRY CAMPUS

Schritt1.1

- eine Datenbankverbindung wurde erzeugt:
  - Die Anwendung ist mit der Datenbank verbunden und sollte in der Lage sein Anfragen an die Datenbank zu schicken
- bevor die Anfragen von der Anwendung an die Datenbank geschickt werden, müssen diese aufbereitet werden
- die Anfragen werden über die vorher erzeugte Verbindung verschickt
  - Aus diesem Grund nutzt man die Verbindung, um die Anfragen aufzubereiten

Connection

- wichtige Methoden:

```
prepareStatement(String query)
statement(String query)                    //eher vermeiden, da unsicher
```

  - Bereitet eine Query vor, um sie an die Datenbank zu schicken
  - Gibt einen PreparedStatement bzw. Statement zurück

SMART INDUSTRY CAMPUS

PreparedStatement

- Wird für vorbereitete SQL-Abfragen genutzt
- sicherer gegen SQL-Injections (vs. Statement)
- Aufbau der Query muss abgewandelt werden:
  - Es werden Platzhalter benötigt – sogenannte Parameter-Marker
  - Diese Platzhalter werden mit „?“ dargestellt, in die später tatsächliche Werte eingesetzt werden
  - Pro Wert wird ein „?“ benötigt
- Beispiel:

```
PreparedStatement p2 = connection.prepareStatement("INSERT INTO Patient(firstname, lastname, email, dayOfBirth, Age, Weight, DoctorID) VALUES (?, ?, ?, ?, ?, ?, ?)");
```

PreparedStatement

- wie setzt man die Werte ein?
- wichtige Methoden:

```
setString(int parameterIndex, String value)
setInt(int parametereIndex, int value)
```

  - Wird genutzt, um die gewünschten Werte in die Query statt den „?“ einzusetzen

SMART INDUSTRY CAMPUS

PreparedStatement

- Beispiel:

```
PreparedStatement p2 = connection.prepareStatement("INSERT INTO Patient(firstname, lastname, email, dayOfBirth, Age, Weight, DoctorID) VALUES (?, ?, ?, ?, ?, ?, ?)");
```

```
PreparedStatement p2 = c.prepareStatement("INSERT INTO Patient(firstname, lastname, email, dayOfBirth, Age, Weight, DoctorID) VALUES (?, ?, ?, ?, ?, ?, ?)");
p2.setString(1, "Lars");
p2.setString(2, "Müller");
p2.setString(3, "Lars.Mueller@web.de");
p2.setDate(4, Date.valueOf(LocalDate.parse("1999-03-14")));
p2.setInt(5, 26);
p2.setInt(6, 26);
p2.setInt(7, 1);
```

Schritt 1.2

- nach der Aufbereitung der Anfragen, in der wir Platzhalter setzen und die Werte sicher einbinden, muss die Anfrage selbst auch verschickt werden
- dieses Verschicken erfolgt durch folgende Befehle

SMART INDUSTRY CAMPUS

PreparedStatement

Methode	Verwendungszweck	Rückgabewert	Beispiel-SQL-Befehl
execute()	Für beliebige SQL-Anfragen, die entweder ein ResultSet oder eine Anzahl der betroffenen Zeilen zurückgeben.	boolean (true/false)	SELECT, UPDATE, DELETE (je nach Rückgabewert)
executeQuery()	Nur für **SELECT**-Abfragen, die ein ResultSet zurückgeben.	ResultSet	SELECT
executeUpdate()	Für **INSERT**, **UPDATE**, und **DELETE**-Befehle.	int (Anzahl der betroffenen Zeilen)	INSERT, UPDATE, DELETE

PreparedStatement

– Beispiel:

```
public static void main(String[] args) {
   try (Connection c = getConnection()) {
      if (c != null) {
         PreparedStatement p2 = c.prepareStatement("INSERT INTO Patient(firstname, lastname, email, dayOfBirth, Age, Weight, DoctorID) VALUES (?, ?, ?, ?, ?, ?, ?)");
         p2.setString(1, "Lars");
         p2.setString(2, "Müller");
         p2.setString(3, "Lars.Mueller@web.de");
         p2.setDate(4, Date.valueOf(LocalDate.parse("1999-03-14")));
         p2.setInt(5, 26);
         p2.setInt(6, 26);
         p2.setInt(7, 1);
         p2.execute();
      }
   } catch (SQLException e) {
      System.out.println(e.getMessage());
   }

}
```

SMART INDUSTRY CAMPUS

Schritt 1.3

- wenn SELECT-Abfragen getätigt werden, resultiert dies oft in einem Ergebnis mit Datensätzen
  - Wenn .executeQuery() ausgeführt wird, erhält man ein ResultSet
  - ResultSet = resultiertes Set an Datensätzen
- dieses Ergebnis möchte die Java-Anwendung verwerten/ nutzen
- dieses Ergebnis repräsentiert oft eine Klasse, die in Java geschrieben wurde
  - Beispiel: Buch, Patient, Autor, …
- um das Ergebnis in eine Klasse zu übersetzen, benötigt es die Klasse ResultSet

ResultSet

- speichert die Ergebnisse einer SQL-Abfrage
- wichtige Methoden:

  next()

  - Verschiebt den Zeiger auf die erste/nächste Zeile aus dem Ergebnis
  - Gibt einen boolean zurück, falls es ein nächstes Element im Set gibt
  - Für Schleifen wichtig!

ResultSet

- speichert die Ergebnisse einer SQL-Abfrage
- wichtige Methoden:

  getString(String columnLabel) / getString(int columnIndex)
  getInt(String columnLabel) / getInt(int columnIndex)

  - Werden verwendet, um den Wert einer bestimmten Spalte aus dem geholten Datensatz abzurufen

ResultSet

- Beispiel:

```
try (Connection c = getConnection()) {
    if (c != null) {
        PreparedStatement p = c.prepareStatement("SELECT * FROM Patient");
        ResultSet resultSet = p.executeQuery();

        while (resultSet.next()) {                    //if(resultSet.next()  bei einem möglichen Ergebnis
            System.out.println(resultSet.getString("firstname"));
            System.out.println(resultSet.getInt("Age"));
        }
    }
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

SMART INDUSTRY CAMPUS

Implementierung mit ORM

- moderne Technologien erlauben es, dass man Datensätze aus der Tabelle direkt in ein objektorientiertes Objekt übersetzt
- dieses Übersetzen nennt man Mapping
- dadurch muss man nicht mehr die einzelnen Spalten- und Zeilenwerte aus dem ResultSet holen
- Moderne Technologien:
  - Hibernate für Java
  - Entity Framework für C#

SMART INDUSTRY CAMPUS

Schritt 2

Schichtenmodell

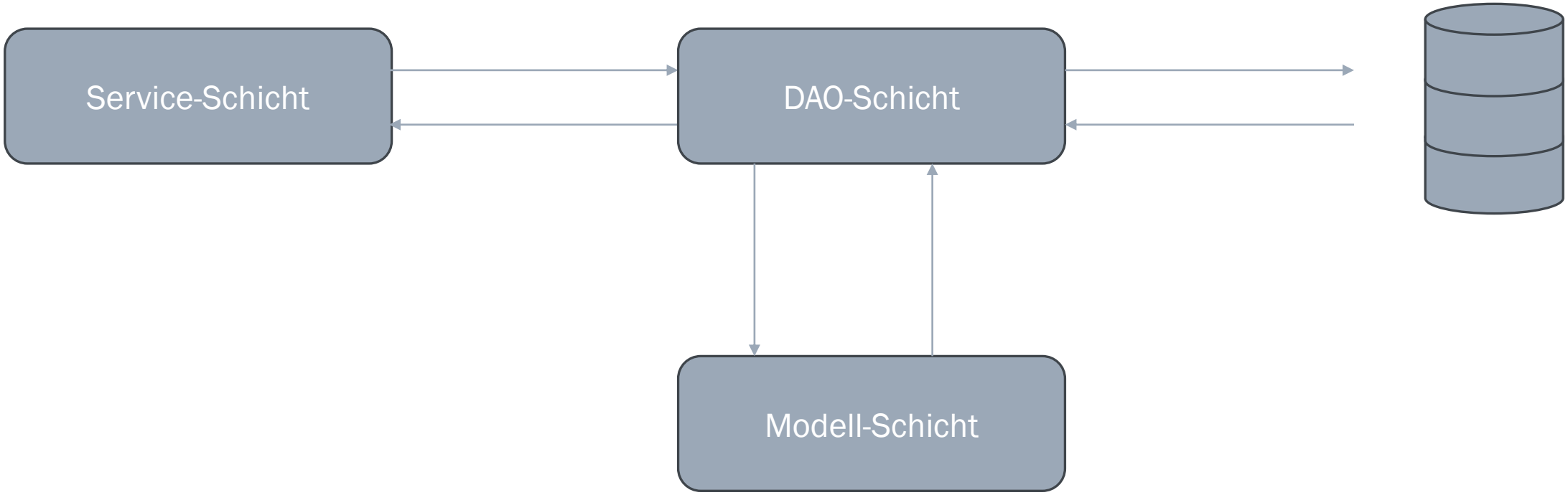
- die Verbindung zur Datenbank und die Anfragen an die Datenbank, die bisher getätigt wurden, werden in der DAO-Schicht festgelegt
  - DAO steht für Data Access Object (Datenzugriffsobjekt)
- die Datenbankanfragen resultieren in einem Ergebnis, das ein Objekt darstellen soll (Buch, Patient, …)
  - Dieses Objekt nennt man auch Model
  - Das Ergebnis wird in ein Model gepackt
- die Anforderung eine Datenbankanfrage zu machen, werden weder vom Model selbst noch von der DAO-Schicht getätigt
  - eine Anwendung greift auf die Service-Schicht zu, die eine Schnittstelle zur DAO bildet

Schichtenmodell

- Modell-Schicht:
  - Repräsentiert eine Entität (z.B. Patient, Book, Author,...) und enthält Attribute sowie grundlegende Validierungen
- Service-Schicht:
  - enthält die Logik und koordinieren die Abläufe zwischen verschiedenen Komponenten
  - Was soll mit den Daten passieren? (Berechnungen, Prüfung, Regeln, ...)
  - Stellt Methoden bereit, die von einem Controller als API genutzt werden können
- DAO (Data Access Object)-Schicht:
  - Ist verantwortlich für den direkten Zugriff auf die Datenbank
  - verbirgt die Datenbank-Logik

Schichtenmodell

– es benötigt eine logische Struktur, um die Anwendung skalierbar, wartbar und sicher zu gestalten



SMART INDUSTRY CAMPUS

Aufgaben

- Erstelle alle Modell-Klassen für die noch nicht erzeugten Tabellen der Library-Bibliothek
- Erstelle alle DAO-Klassen für die noch nicht erzeugten Tabellen der Library-Bibliothek
- Erstelle alle Service-Klassen für die noch nicht erzeugten Tabellen der Library-Bibliothek