Practical Assessment: CRUD Operations in Java using JDBC & ORM

Instructions:

- Each question requires a fully functional Java program.
- Use JDBC for direct database interactions and ORM (like Hibernate) for object-relational mapping.
- Ensure database connectivity is correctly configured.
- Write clean and modular code following best practices.

1. Implement a Student Management System using JDBC

Create a **Java console-based** application using **JDBC** that allows users to **perform CRUD operations on a "students" table** in a MySQL database. The application should allow:

1. Adding a new student with id, name, email, and course.
2. Updating student details by id.
3. Deleting a student record by id.
4. Displaying all students.
5. Searching for a student by id.

2. Implement Employee Management System using Hibernate ORM

Design and implement an **Employee Management System** using **Hibernate ORM**. The system should allow:

1. Adding a new employee with fields: employeeId, name, department, salary.
2. Fetching and displaying all employee records.
3. Updating the salary of an employee based on employeeId.
4. Deleting an employee record using Hibernate.
5. Implement proper annotations for Hibernate Entity class.

3. JDBC Transaction Handling for Bank Account Transfers

Write a Java program using **JDBC transactions** to **transfer money between two bank accounts**. The program should:

1. Accept senderId, receiverId, and amount.
2. Deduct the amount from senderId and add it to receiverId.
3. Ensure atomicity using JDBC **commit and rollback** in case of failure.
4. Handle exceptions like **insufficient balance, incorrect account ID**.

4. Bookstore Management System using JDBC

Create a Java program that manages a **Bookstore** using **JDBC** with the following features:

1. Add a book with bookId, title, author, price.
2. Display all books available.
3. Update the price of a book based on bookId.
4. Delete a book from the catalog.
5. Search for books by author or title.

5. Library Management System using Hibernate ORM

Develop a Hibernate-based **Library Management System** where:

1. Book is an entity with bookId, title, author, availability.
2. Member is an entity with memberId, name, email.
3. A member can borrow a book (update availability).
4. Return a book (update availability).
5. Display all borrowed books with borrower details using **Hibernate Query Language (HQL)**.

6. Implement CRUD Operations using DAO Pattern with JDBC

Create a Java program that uses the **DAO (Data Access Object) Pattern** for managing Product data using **JDBC**. The program should:

1. Implement a ProductDAO interface with methods: addProduct(), updateProduct(), deleteProduct(), getProductById(), getAllProducts().
2. Implement ProductDAOImpl class using **PreparedStatement** and **ResultSet**.
3. Use **MySQL Database** for storing product details.

7. Implement Many-to-One Relationship using Hibernate

Consider a scenario where multiple Employees belong to a Department. Implement this Many-to-One relationship using Hibernate ORM:

1. Create Department entity with deptId, deptName.
2. Create Employee entity with employeeId, name, salary, and a foreign key deptId.
3. Fetch all employees under a specific department using **HQL Query**.

8. Create a Customer Order Management System using JDBC

Write a JDBC program for an **Order Management System** with tables:

1. **Customers** (customerId, name, email).
2. **Orders** (orderId, customerId, product, quantity).
3. Implement methods to:
    - Place an order (insert into Orders table).
    - Fetch all orders of a specific customer using **JOIN queries**.
    - Cancel an order.

9. Implement Pagination in Hibernate for Large Data Sets

You have a Users table containing thousands of records. Write a Hibernate-based Java program to implement **pagination** using Query.setFirstResult() and Query.setMaxResults(). The program should:

1. Display **10 records per page**.
2. Allow users to navigate **next** and **previous pages**.
3. Use **Criteria API** for fetching data efficiently.

10. Implement Login System with JDBC and Password Hashing

Develop a **secure login system** using **JDBC and password hashing**. The program should:

1. Allow users to register with username and hashed password (using SHA-256 or bcrypt).
2. Validate login credentials by verifying the password against the hashed value stored in the database.
3. Prevent SQL Injection attacks using **PreparedStatement**.