MODULE: Java Programming

ASSIGNMENT 11 – JDBC CRUD + 3-Tier Architecture + DAO Layer Design

Learning Objectives

After completing this assignment, the participant will be able to:

- Design a book-store application using the **DAO Pattern**
- Implement **CRUD operations using JDBC**
- Build a **3-Tier Architecture** (Web → Service → Persistence)
- Understand exception wrapping & rethrowing in DAO layer
- Keep persistence design **flexible** for future migration to Hibernate/JPA
- Apply good coding practices such as interface-based programming, loose coupling, and abstraction

General Instructions

1. Follow the **3-tier architecture** strictly:
    - **Web Layer / Presentation Layer** → Simulated using a CLI or simple UI
    - **Service Layer** → Contains business logic
    - **DAO Layer** → JDBC logic only
2. Your code must be designed so that **switching JDBC → Hibernate later requires minimal changes**.
3. Throw custom exceptions from DAO layer using a wrapper (`DaoException`).
4. Your JDBC code **must use PreparedStatement**, not Statement.
5. Use a MySQL / PostgreSQL / Oracle DB (your choice).
6. Create proper SQL table before coding:

```
CREATE TABLE books (
    id INT PRIMARY KEY AUTO_INCREMENT,
    isbn VARCHAR(20),
    title VARCHAR(200),
    author VARCHAR(100),
    price DOUBLE
);
```

7. Test your CRUD operations using at least 5 sample books.

Estimated Time

Section	Time
DAO + Interface Design	45–60 min
JDBC CRUD Implementation	60–90 min
Service Layer	30 min
Testing + Exception Handling	30 min

Evaluation Rubric

Criteria	Weight
Correct JDBC CRUD Implementation	40%
Clean Architecture (3-tier + DAO)	25%
Exception Handling Quality	15%
Test Cases & Output	10%
Code Style & Maintainability	10%

Business Requirement (As Provided)

Create a Book Store Application for **BPB Publications**.

The application must support:

- Add a Book
- Edit a Book
- Delete a Book
- Find a specific Book
- List all Books

Initial technology: **JDBC**
Future technology: **Hibernate ORM**

Your architecture must allow switching persistence layer **without modifying service or presentation layers**.

Starter Code (Given in Requirement)

Use and extend the following:

```
public class Book {
    private int id;
    private String isbn;
    private String title;
    private String author;
    private double price;
    // getter setter, constructor
}
```

DAO Interface

```
public interface BookDao {
    public List<Book> getAllBooks() throws DaoException;
    public Book addBook(Book book) throws DaoException;
    public void deleteBook(int id) throws DaoException;
    public void updateBook(int id, Book book) throws DaoException;
    public Book getBookById(int id) throws DaoException;
```

```
}
```

DAO Implementation

```
public class BookDaoImp implements BookDao {
    // implement JDBC CRUD operations
}
```

Custom DAO Exception

```
public class DaoException extends Exception {
    public DaoException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

REQUIRED TASKS

Q1. Create the DAO Layer (Persistence Layer)

Implement **BookDaoImp** with the following JDBC operations:

1. `addBook(Book book)`
2. `getAllBooks()`
3. `getBookById(int id)`
4. `updateBook(int id, Book book)`
5. `deleteBook(int id)`

Mandatory Requirements

- Use `PreparedStatement`.
- Manage connections using a utility class: `DBUtil.getConnection()`.
- Use **try-with-resources**.
- Wrap SQL exceptions inside `DaoException`.

Q2. Create the Service Layer

Create:

```
public class BookService {
    private BookDao bookDao;
}
```

Service layer responsibilities:

- Validate business rules (e.g., price > 0, title not empty)
- Call DAO methods
- Convert SQL errors to user-friendly messages

This layer must **not contain JDBC code**.

Q3. Create the Presentation Layer

You may use:

- Console UI (Scanner-based)
- Menu-driven interface

Menu example:

```
1. Add Book
2. Edit Book
3. Delete Book
4. Search Book
5. List Books
6. Exit
```

Each option must call the appropriate **service layer** method.

Q4. Implement Exception Wrapping and Rethrowing

Research (as the assignment requires):

✔ "Why DAO Exceptions must wrap SQLExceptions?"

✔ "Why exceptions should be rethrown instead of swallowed?"

Then implement:

- A wrapper exception `DaoException`
- Throw it when JDBC operations fail
- Let upper layers decide how to handle it

Q5. Test the Application

Insert at least **5 records** of your favorite books.
Example suggestions:

- Head First Java
- Effective Java
- Clean Code
- Java Concurrency in Practice
- Spring in Action

Test each CRUD operation thoroughly.

BONUS TASKS (Optional But Recommended)

Bonus 1

Add **search by title** and **search by author** using LIKE queries.

Bonus 2

Add pagination feature:
`List<Book> getBooks(int page, int size)`

Bonus 3

Prepare the application for Hibernate migration:

- Create a `BookDaoHibernateImp` skeleton class
- Implement only method signatures

Bonus 4

Add logging using Java's logging or Log4j.

Reflection Questions

1. Why is 3-tier architecture preferred over tightly-coupled monolithic code?
2. How does the DAO pattern help in switching technologies (JDBC → Hibernate)?
3. Why must DAO layer wrap SQLExceptions in custom exceptions?
4. What design changes will you need when migrating to Hibernate?
5. Which CRUD operation did you find most challenging and why?