MODULE: Java Programming

ASSIGNMENT 12 – Unit Testing with JUnit & Log4j2

(Participants must refer to the existing Bookstore Application Service you created earlier.)

Learning Objectives

After completing this lab, participants will be able to:

- Write unit tests using **JUnit (4 or 5)**
- Understand test lifecycle (`@BeforeEach`, `@AfterEach`, `@BeforeAll`, `@AfterAll`)
- Apply **assertions** to validate business logic
- Mock dependencies (optional advanced task)
- Use **Log4j2** for logging inside tests
- Improve confidence and reliability of service-layer methods
- Follow best practices for professional testing in enterprise applications

General Instructions

1. You must write unit tests for the **Bookstore Service Layer** created in previous assignments (Assignment 11 – JDBC + 3 Tier).
2. Do **NOT** use JDBC or database calls directly in tests.
3. All tests must be placed inside a standard folder structure:

```
src/test/java/com/app/service/
```

4. Use **Log4j2** for logging test execution steps (info, error).
5. Each test method name must follow clear naming conventions such as:

```
shouldAddBookSuccessfully()
shouldThrowExceptionWhenBookNotFound()
```

6. Write separate tests for **positive and negative scenarios**.
7. You must have **at least 10 test cases** to pass the assignment.

Estimated Time

Task	Time
Setting up JUnit & Log4j2	20–30 min
Writing service tests	45–60 min

Task	Time
Running & fixing failing tests	20 min

Evaluation Rubric

Criteria	Weight
Test coverage & completeness	40%
Correct use of JUnit annotations	20%
Logging with Log4j2	15%
Quality of assertions	15%
Code style & naming	10%

BOOKSTORE SERVICE UNDER TEST

You must write tests for:

```
BookService
```

Typical methods include:

```
addBook(Book book)
updateBook(int id, Book book)
deleteBook(int id)
getBookById(int id)
getAllBooks()
```

If your exact method signatures differ, adapt accordingly.

Q1. Setup JUnit Test Class for BookService

Tasks

1. Create a test class:

```
BookServiceTest
```

2. Use JUnit 4 or 5 (depending on your setup).

3. Use annotations:

    - `@BeforeAll`
    - `@AfterAll`
    - `@BeforeEach`
    - `@AfterEach`

Expected Outcome

Your test environment should initialize BookService and reset it for every test.

Q2. Write Test Cases for addBook()

Write at least two tests:

1. **Positive test**

    - Adding a valid book should increase count.

- Use assertions like:

```
assertEquals(expected, actual);
```

2. **Negative test**
    - Adding a book with invalid price or empty title should throw exception.

Logging

Use Log4j2 to log test start and result:

```
logger.info("Starting addBook positive test...");
```

Q3. Write Test Cases for getBookById()

Tasks

- Test valid ID → Book object returned
- Test invalid ID → should throw custom exception or return null (your design choice)

Assertions to use:

- `assertNotNull`
- `assertThrows` (JUnit 5)

Q4. Write Test Cases for updateBook()

Tasks

- Update an existing book and verify updated fields
- Try updating with invalid data (null title, negative price)

Assertions:

```
assertEquals()
assertNotEquals()
```

Q5. Write Test Cases for deleteBook()

Tasks

- Delete an existing book → verify count is reduced
- Delete a non-existing ID → verify correct exception/logging

Log the outcome:

```
logger.error("Book not found for deletion");
```

Q6. Write Test Cases for getAllBooks()

Tasks

- Add some sample books
- Retrieve list and verify size
- Verify ordering (if applicable)

OPTIONAL ADVANCED TASKS

Bonus 1 — Use Mockito (if allowed)

Mock DAO layer and test BookService independently.

Bonus 2 — Parametrized Tests

Use JUnit parameterized tests for repetitive scenarios such as invalid book price.

Bonus 3 — Test Log4j2 Output

Redirect console logs to file and verify structured logs.

Bonus 4 — Coverage Report

Generate coverage using JaCoCo.

Reflection Questions

1. Why is unit testing essential before integrating with JDBC or Hibernate?
2. What kind of bugs were detected only through tests?
3. What makes a test *good* or *bad*?
4. Why is mocking useful in service-layer testing?
5. How does logging improve debugging within tests?