**Day3\_Labs: Mocking Repository Layer & Testing Service Layer in Java with Mockito**

**Objective**

* Learn how to mock a repository layer using **Mockito**.
* Write unit tests for a **Service layer** that depends on the repository.
* Verify method calls, return values, and exception handling.

**Project Setup**

1. Create a **Maven project** named student-management.
2. Add dependencies in pom.xml:

<dependencies>

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.8.0</version>

<scope>test</scope>

</dependency>

<!-- For using @ExtendWith(MockitoExtension.class) -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<version>5.8.0</version>

<scope>test</scope>

</dependency>

</dependencies>

Run:

mvn clean install

**Step 1 – Define the Domain Model**

src/main/java/com/example/studentmanagement/model/Student.java

package com.example.studentmanagement.model;

public class Student {

private int id;

private String name;

private double grade;

public Student(int id, String name, double grade) {

this.id = id;

this.name = name;

this.grade = grade;

}

public int getId() { return id; }

public String getName() { return name; }

public double getGrade() { return grade; }

}

**Step 2 – Define the Repository Contract**

src/main/java/com/example/studentmanagement/repository/StudentRepository.java

package com.example.studentmanagement.repository;

import com.example.studentmanagement.model.Student;

import java.util.List;

public interface StudentRepository {

Student getStudentById(int id);

List<Student> getAllStudents();

void addStudent(Student student);

}

**Step 3 – Create the Service Layer**

src/main/java/com/example/studentmanagement/service/StudentService.java

package com.example.studentmanagement.service;

import com.example.studentmanagement.model.Student;

import com.example.studentmanagement.repository.StudentRepository;

import java.util.List;

import java.util.OptionalDouble;

public class StudentService {

private final StudentRepository repository;

public StudentService(StudentRepository repository) {

this.repository = repository;

}

public Student getStudent(int id) {

Student student = repository.getStudentById(id);

if (student == null) {

throw new RuntimeException("Student not found");

}

return student;

}

public double getAverageGrade() {

List<Student> students = repository.getAllStudents();

if (students.isEmpty()) {

return 0;

}

OptionalDouble avg = students.stream().mapToDouble(Student::getGrade).average();

return avg.orElse(0);

}

public void registerStudent(Student student) {

if (student.getName() == null || student.getName().isBlank()) {

throw new IllegalArgumentException("Student name is required");

}

repository.addStudent(student);

}

}

**Step 4 – Write Unit Tests with JUnit 5 + Mockito**

src/test/java/com/example/studentmanagement/service/StudentServiceTest.java

package com.example.studentmanagement.service;

import com.example.studentmanagement.model.Student;

import com.example.studentmanagement.repository.StudentRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.\*;

import java.util.Arrays;

import java.util.Collections;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

class StudentServiceTest {

//Mock Repository

//Inject Service

@Test

void getStudent\_existingId\_returnsStudent() {

// Arrange

// Act

// Assert

}

@Test

void getStudent\_nonExistingId\_throwsException() {

}

@Test

void getAverageGrade\_withStudents\_returnsCorrectAverage() {

}

@Test

void registerStudent\_emptyName\_throwsException() {

}

@Test

void getAverageGrade\_noStudents\_returnsZero() {

}

}

**Lab Tasks for additional practice**

1. **Run the tests** and ensure all pass
2. Add a new method getTopStudent() in the service that returns the student with the highest grade.
   * Write **unit tests** for it.
3. Add edge-case tests (e.g., null input student, empty repository list).

At the end of this lab, students will have practiced **mocking repository layer with Mockito** and **testing service layer with JUnit 5**.