**Verifying Interactions with Mockito**

**1. Scenario Setup**

We'll test a SeatReservationService that interacts with a SeatMapRepository.

**1.1 Create the Repository Interface**

public interface SeatMapRepository {

boolean reserveSeat(String flightNumber, int row, char seat);

boolean isSeatAvailable(String flightNumber, int row, char seat);

List<String> getReservedSeats(String flightNumber);

}

**1.2 Create the Service to Test**

public class SeatReservationService {

private final SeatMapRepository seatMapRepo;

public SeatReservationService(SeatMapRepository seatMapRepo) {

this.seatMapRepo = seatMapRepo;

}

public ReservationResult reserveSeat(String flightNumber, int row, char seat, String passengerId) {

if (!seatMapRepo.isSeatAvailable(flightNumber, row, seat)) {

return ReservationResult.seatUnavailable();

}

boolean success = seatMapRepo.reserveSeat(flightNumber, row, seat);

if (success) {

return ReservationResult.success(passengerId, flightNumber, row, seat);

} else {

return ReservationResult.failure();

}

}

public boolean bulkReserve(String flightNumber, List<SeatRequest> requests) {

for (SeatRequest request : requests) {

if (!seatMapRepo.reserveSeat(flightNumber, request.row(), request.seat())) {

return false;

}

}

return true;

}

}

record SeatRequest(int row, char seat) {}

record ReservationResult(String status, String details) {

public static ReservationResult success(String passengerId, String flight, int row, char seat) {

return new ReservationResult("CONFIRMED",

String.format("%s-%d%c for %s", flight, row, seat, passengerId));

}

public static ReservationResult seatUnavailable() {

return new ReservationResult("FAILED", "Seat unavailable");

}

public static ReservationResult failure() {

return new ReservationResult("FAILED", "Reservation failed");

}

}

**2. Test Implementation with Verification**

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.ArgumentCaptor;

import org.mockito.Captor;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import static org.mockito.Mockito.\*;

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

@ExtendWith(MockitoExtension.class)

class SeatReservationServiceTest {

@Mock

private SeatMapRepository seatMapRepo;

@InjectMocks

private SeatReservationService reservationService;

@Captor

private ArgumentCaptor<Character> seatCaptor;

@Test

void reserveSeat\_shouldVerifyRepositoryInteractions() {

// Arrange

String testFlight = "FL123";

when(seatMapRepo.isSeatAvailable(testFlight, 12, 'B')).thenReturn(true);

when(seatMapRepo.reserveSeat(testFlight, 12, 'B')).thenReturn(true);

// Act

ReservationResult result = reservationService.reserveSeat(testFlight, 12, 'B', "PAX001");

// Assert

assertEquals("CONFIRMED", result.status());

// Verify interactions

verify(seatMapRepo).isSeatAvailable(testFlight, 12, 'B');

verify(seatMapRepo).reserveSeat(testFlight, 12, 'B');

// Verify no other interactions

verifyNoMoreInteractions(seatMapRepo);

}

@Test

void bulkReserve\_shouldVerifyMultipleCalls() {

// Arrange

String testFlight = "FL456";

List<SeatRequest> requests = List.of(

new SeatRequest(10, 'A'),

new SeatRequest(10, 'B'),

new SeatRequest(11, 'A')

);

when(seatMapRepo.reserveSeat(eq(testFlight), anyInt(), anyChar()))

.thenReturn(true);

// Act

boolean success = reservationService.bulkReserve(testFlight, requests);

// Assert

assertTrue(success);

// Verify each seat was reserved

verify(seatMapRepo, times(3)).reserveSeat(eq(testFlight), anyInt(), anyChar());

// Verify specific seat letters were used

verify(seatMapRepo).reserveSeat(testFlight, 10, 'A');

verify(seatMapRepo).reserveSeat(testFlight, 10, 'B');

verify(seatMapRepo).reserveSeat(testFlight, 11, 'A');

}

@Test

void reserveSeat\_shouldCaptureSeatArgument() {

// Arrange

String testFlight = "FL789";

when(seatMapRepo.isSeatAvailable(eq(testFlight), eq(15), seatCaptor.capture()))

.thenReturn(true);

when(seatMapRepo.reserveSeat(eq(testFlight), eq(15), anyChar()))

.thenReturn(true);

// Act

reservationService.reserveSeat(testFlight, 15, 'C', "PAX002");

// Assert

assertEquals('C', seatCaptor.getValue());

}

}

**Key Verification Techniques**

**1. Basic Verification**

verify(mock).method(args); // Called once

verify(mock, times(n)).method(args); // Called exactly n times

verify(mock, atLeastOnce()).method(args);

verify(mock, never()).method(args); // Never called

**2. Argument Matchers**

verify(repo).reserveSeat(eq("FL123"), anyInt(), eq('B'));

// eq() for specific values

// any() for any value of that type

**3. Verification Modes**

InOrder inOrder = inOrder(mock1, mock2);

inOrder.verify(mock1).method1();

inOrder.verify(mock2).method2();

**4. Argument Captors**

@Captor ArgumentCaptor<String> stringCaptor;

verify(mock).method(stringCaptor.capture());

assertEquals("expected", stringCaptor.getValue());