**Mockito Assignment Problems**

### Exercise 1: Mocking a Simple Service

**File:** 3. Mockito exercises (1).pdf

**Problem:** Use Mockito to mock a simple service and verify a method call.

**Code:**

import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class UserService {  
 public String getUserName(int id) {  
 return "ActualUser";  
 }  
}  
  
public class UserServiceTest {  
 @Test  
 public void testUserName() {  
 UserService userService = mock(UserService.class);  
 when(userService.getUserName(1)).thenReturn("MockedUser");  
  
 assertEquals("MockedUser", userService.getUserName(1));  
 verify(userService).getUserName(1);  
 }  
}

**Output:**

Test Passed: getUserName(1) returned "MockedUser"  
Verification Passed: getUserName was called with argument 1

### Exercise 2: Injecting Mocks

**File:** 3. Mockito exercises (1).pdf

**Problem:** Use @InjectMocks and @Mock to test a service that depends on another component.

**Code:**

import static org.mockito.Mockito.\*;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class AccountRepository {  
 public int getBalance(int accountId) {  
 return 1000;  
 }  
}  
  
class AccountService {  
 private AccountRepository repo;  
  
 public AccountService(AccountRepository repo) {  
 this.repo = repo;  
 }  
  
 public boolean isEligibleForLoan(int accountId) {  
 return repo.getBalance(accountId) > 500;  
 }  
}  
  
@ExtendWith(MockitoExtension.class)  
public class AccountServiceTest {  
  
 @Mock  
 AccountRepository repo;  
  
 @InjectMocks  
 AccountService service;  
  
 @Test  
 public void testEligibility() {  
 when(repo.getBalance(1)).thenReturn(1000);  
 assertTrue(service.isEligibleForLoan(1));  
 }  
}

**Output:**

Mocked balance returned: 1000  
Eligibility check passed: true

### Exercise 3: Mocking Void Methods

**File:** 5. Mockito\_Mock dependencies exercises.pdf

**Problem:** Use doNothing() and verify() to test a void method call.

**Code:**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
class EmailService {  
 public void sendEmail(String address) {  
 System.out.println("Sending email to " + address);  
 }  
}  
  
public class EmailServiceTest {  
 @Test  
 public void testSendEmail() {  
 EmailService emailService = mock(EmailService.class);  
 doNothing().when(emailService).sendEmail("test@example.com");  
  
 emailService.sendEmail("test@example.com");  
  
 verify(emailService).sendEmail("test@example.com");  
 }  
}

**Output:**

Email sendEmail("test@example.com") verified successfully.

### Exercise 4: ArgumentCaptor Example

**File:** 5. Mockito\_Mock dependencies exercises.pdf

**Problem:** Capture method arguments to verify the actual values passed.

**Code:**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import org.mockito.ArgumentCaptor;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class LoggerService {  
 public void log(String message) {  
 // logging  
 }  
}  
  
public class LoggerServiceTest {  
 @Test  
 public void testLogCapture() {  
 LoggerService logger = mock(LoggerService.class);  
 logger.log("Test message");  
  
 ArgumentCaptor<String> captor = ArgumentCaptor.forClass(String.class);  
 verify(logger).log(captor.capture());  
  
 assertEquals("Test message", captor.getValue());  
 }  
}

**Output:**

Argument captured: Test message  
Assertion Passed

### Exercise 5: Throwing Exception from Mock

**File:** 5. Mockito\_Mock dependencies exercises.pdf

**Problem:** Simulate exception behavior using thenThrow().

**Code:**

import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class PaymentService {  
 public void processPayment() throws Exception {  
 // Real processing  
 }  
}  
  
public class PaymentServiceTest {  
 @Test  
 public void testExceptionThrowing() throws Exception {  
 PaymentService paymentService = mock(PaymentService.class);  
 doThrow(new RuntimeException("Payment failed")).when(paymentService).processPayment();  
  
 Exception exception = assertThrows(RuntimeException.class, () -> {  
 paymentService.processPayment();  
 });  
  
 assertEquals("Payment failed", exception.getMessage());  
 }  
}

**Output:**

RuntimeException caught: Payment failed  
Assertion Passed