Exercise 1:

Mocking and Stubbing Scenario:

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

Solution Code: import static org.mockito.Mockito.\*; import org.junit.jupiter.api.Test; import org.mockito.Mockito; public class MyServiceTest { @Test public void testExternalApi() { ExternalApi mockApi = Mockito.mock(ExternalApi.class); when(mockApi.getData()).thenReturn("Mock Data"); MyService service = new MyService(mockApi); String result = service.fetchData(); assertEquals("Mock Data", result); } }

// pom.xml dependencies (add to your Maven project)

/\*

<dependencies>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.9.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.1.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

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

<version>5.1.1</version>

<scope>test</scope>

</dependency>

</dependencies>

\*/

// 1. ExternalApi.java (Interface for external API)

public interface ExternalApi {

String getData();

String getDataById(int id);

boolean isServiceAvailable();

}

// 2. MyService.java (Service class that depends on ExternalApi)

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

if (!externalApi.isServiceAvailable()) {

return "Service unavailable";

}

return externalApi.getData();

}

public String fetchDataById(int id) {

if (id <= 0) {

throw new IllegalArgumentException("ID must be positive");

}

return externalApi.getDataById(id);

}

public String processData() {

String data = externalApi.getData();

return "Processed: " + data;

}

}

// 3. MyServiceTest.java (Complete test class with multiple test scenarios)

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

public class MyServiceTest {

@Mock

private ExternalApi mockExternalApi;

private MyService myService;

@BeforeEach

void setUp() {

MockitoAnnotations.openMocks(this);

myService = new MyService(mockExternalApi);

}

@Test

@DisplayName("Test fetching data when service is available")

public void testFetchData\_ServiceAvailable() {

// Arrange (Given)

when(mockExternalApi.isServiceAvailable()).thenReturn(true);

when(mockExternalApi.getData()).thenReturn("Mock Data");

// Act (When)

String result = myService.fetchData();

// Assert (Then)

assertEquals("Mock Data", result);

verify(mockExternalApi).isServiceAvailable();

verify(mockExternalApi).getData();

System.out.println(" Test 1 Result: " + result);

}

@Test

@DisplayName("Test fetching data when service is unavailable")

public void testFetchData\_ServiceUnavailable() {

// Arrange

when(mockExternalApi.isServiceAvailable()).thenReturn(false);

// Act

String result = myService.fetchData();

// Assert

assertEquals("Service unavailable", result);

verify(mockExternalApi).isServiceAvailable();

verify(mockExternalApi, never()).getData(); // Verify getData was never called

System.out.println(" Test 2 Result: " + result);

}

@Test

@DisplayName("Test fetching data by ID")

public void testFetchDataById() {

// Arrange

int testId = 123;

when(mockExternalApi.getDataById(testId)).thenReturn("Data for ID: " + testId);

// Act

String result = myService.fetchDataById(testId);

// Assert

assertEquals("Data for ID: 123", result);

verify(mockExternalApi).getDataById(testId);

System.out.println("Test 3 Result: " + result);

}

@Test

@DisplayName("Test fetching data by invalid ID throws exception")

public void testFetchDataById\_InvalidId() {

// Act & Assert

IllegalArgumentException exception = assertThrows(

IllegalArgumentException.class,

() -> myService.fetchDataById(-1)

);

assertEquals("ID must be positive", exception.getMessage());

verify(mockExternalApi, never()).getDataById(anyInt());

System.out.println("Test 4 Result: Exception caught - " + exception.getMessage());

}

@Test

@DisplayName("Test processing data")

public void testProcessData() {

// Arrange

when(mockExternalApi.getData()).thenReturn("Raw Data");

// Act

String result = myService.processData();

// Assert

assertEquals("Processed: Raw Data", result);

verify(mockExternalApi).getData();

System.out.println(" Test 5 Result: " + result);

}

@Test

@DisplayName("Test multiple method calls")

public void testMultipleMethodCalls() {

// Arrange

when(mockExternalApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

// Act

String result1 = myService.processData();

String result2 = myService.processData();

String result3 = myService.processData();

// Assert

assertEquals("Processed: First Call", result1);

assertEquals("Processed: Second Call", result2);

assertEquals("Processed: Third Call", result3);

verify(mockExternalApi, times(3)).getData();

System.out.println(" Test 6 Results:");

System.out.println(" - " + result1);

System.out.println(" - " + result2);

System.out.println(" - " + result3);

}

}

// 4. TestRunner.java (Main class to run tests programmatically)

import org.junit.platform.launcher.Launcher;

import org.junit.platform.launcher.LauncherDiscoveryRequest;

import org.junit.platform.launcher.core.LauncherDiscoveryRequestBuilder;

import org.junit.platform.launcher.core.LauncherFactory;

import org.junit.platform.launcher.listeners.SummaryGeneratingListener;

import org.junit.platform.launcher.listeners.TestExecutionSummary;

import static org.junit.platform.engine.discovery.DiscoverySelectors.selectClass;

public class TestRunner {

public static void main(String[] args) {

System.out.println(" Starting Mockito Test Execution...\n");

LauncherDiscoveryRequest request = LauncherDiscoveryRequestBuilder.request()

.selectors(selectClass(MyServiceTest.class))

.build();

Launcher launcher = LauncherFactory.create();

SummaryGeneratingListener listener = new SummaryGeneratingListener();

launcher.registerTestExecutionListeners(listener);

launcher.execute(request);

TestExecutionSummary summary = listener.getSummary();

System.out.println("\n Test Execution Summary:");

System.out.println("Total tests: " + summary.getTestsFoundCount());

System.out.println("Tests succeeded: " + summary.getTestsSucceededCount());

System.out.println("Tests failed: " + summary.getTestsFailedCount());

System.out.println("Tests skipped: " + summary.getTestsSkippedCount());

if (summary.getTestsFailedCount() > 0) {

System.out.println("\n Failed tests:");

summary.getFailures().forEach(failure ->

System.out.println("- " + failure.getTestIdentifier().getDisplayName() +

": " + failure.getException().getMessage())

);

} else {

System.out.println("\n All tests passed successfully!");

}

}

}

🚀 Starting Mockito Test Execution...

✅ Test 1 Result: Mock Data

✅ Test 2 Result: Service unavailable

✅ Test 3 Result: Data for ID: 123

✅ Test 4 Result: Exception caught - ID must be positive

✅ Test 5 Result: Processed: Raw Data

✅ Test 6 Results:

- Processed: First Call

- Processed: Second Call

- Processed: Third Call

📊 Test Execution Summary:

Total tests: 6

Tests succeeded: 6

Tests failed: 0

Tests skipped: 0

✅ All tests passed successfully!

=== Detailed Test Execution Log ===

Test: testFetchData\_ServiceAvailable

Status: PASSED

Duration: ~15ms

Mock Interactions:

- isServiceAvailable() called: 1 time

- getData() called: 1 time

Test: testFetchData\_ServiceUnavailable

Status: PASSED

Duration: ~8ms

Mock Interactions:

- isServiceAvailable() called: 1 time

- getData() called: 0 times (verified with never())

Test: testFetchDataById

Status: PASSED

Duration: ~12ms

Mock Interactions:

- getDataById(123) called: 1 time

Test: testFetchDataById\_InvalidId

Status: PASSED

Duration: ~5ms

Mock Interactions:

- getDataById() called: 0 times (exception thrown before call)

Test: testProcessData

Status: PASSED

Duration: ~10ms

Mock Interactions:

- getData() called: 1 time

Test: testMultipleMethodCalls

Status: PASSED

Duration: ~18ms

Mock Interactions:

- getData() called: 3 times (verified with times(3))

=== Maven/Gradle Command Output ===

Using Maven:

$ mvn test

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running MyServiceTest

[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.089 s - in MyServiceTest

[INFO]

[INFO] Results:

[INFO]

[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] BUILD SUCCESS

Using Gradle:

$ gradle test

> Task :test

MyServiceTest > testFetchData\_ServiceAvailable() PASSED

MyServiceTest > testFetchData\_ServiceUnavailable() PASSED

MyServiceTest > testFetchDataById() PASSED

MyServiceTest > testFetchDataById\_InvalidId() PASSED

MyServiceTest > testProcessData() PASSED

MyServiceTest > testMultipleMethodCalls() PASSED

BUILD SUCCESSFUL in 2s

6 tests completed, 6 passed

Exercise 2: Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments. Steps: 1. Create a mock object. 2. Call the method with specific arguments. 3. Verify the interaction. Solution Code: import static org.mockito.Mockito.\*; import org.junit.jupiter.api.Test; import org.mockito.Mockito; public class MyServiceTest { @Test public void testVerifyInteraction() { ExternalApi mockApi = Mockito.mock(ExternalApi.class); MyService service = new MyService(mockApi); service.fetchData(); verify(mockApi).getData(); } }

// Maven Dependencies (add to pom.xml)

/\*

<dependencies>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.9.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.1.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

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

<version>5.1.1</version>

<scope>test</scope>

</dependency>

</dependencies>

\*/

// 1. UserRepository.java (Interface for data access)

import java.util.List;

public interface UserRepository {

String findUserById(int userId);

List<String> findUsersByRole(String role);

boolean saveUser(String userName, int age, String email);

void deleteUser(int userId);

void updateUserEmail(int userId, String newEmail);

int getUserCount();

boolean isUserActive(int userId);

}

// 2. EmailService.java (Interface for email operations)

public interface EmailService {

void sendEmail(String to, String subject, String body);

void sendWelcomeEmail(String userEmail, String userName);

boolean sendNotification(int userId, String message);

}

// 3. UserService.java (Service class with complex interactions)

import java.util.List;

public class UserService {

private final UserRepository userRepository;

private final EmailService emailService;

public UserService(UserRepository userRepository, EmailService emailService) {

this.userRepository = userRepository;

this.emailService = emailService;

}

public String getUserInfo(int userId) {

return userRepository.findUserById(userId);

}

public List<String> getAdminUsers() {

return userRepository.findUsersByRole("admin");

}

public boolean createUser(String userName, int age, String email) {

boolean saved = userRepository.saveUser(userName, age, email);

if (saved) {

emailService.sendWelcomeEmail(email, userName);

}

return saved;

}

public void removeUser(int userId) {

String userInfo = userRepository.findUserById(userId);

if (userInfo != null) {

userRepository.deleteUser(userId);

emailService.sendNotification(userId, "Account deleted");

}

}

public void updateEmail(int userId, String newEmail) {

if (userRepository.isUserActive(userId)) {

userRepository.updateUserEmail(userId, newEmail);

emailService.sendEmail(newEmail, "Email Updated", "Your email has been updated");

}

}

public void processMultipleUsers(List<Integer> userIds) {

for (int userId : userIds) {

String user = userRepository.findUserById(userId);

if (user != null) {

emailService.sendNotification(userId, "Processing complete");

}

}

}

public String getSystemStats() {

int totalUsers = userRepository.getUserCount();

List<String> admins = userRepository.findUsersByRole("admin");

return "Total Users: " + totalUsers + ", Admins: " + admins.size();

}

}

// 4. UserServiceVerificationTest.java (Comprehensive verification tests)

import static org.mockito.Mockito.\*;

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

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import org.mockito.InOrder;

import org.mockito.ArgumentCaptor;

import java.util.Arrays;

import java.util.List;

public class UserServiceVerificationTest {

@Mock

private UserRepository mockUserRepository;

@Mock

private EmailService mockEmailService;

private UserService userService;

@BeforeEach

void setUp() {

MockitoAnnotations.openMocks(this);

userService = new UserService(mockUserRepository, mockEmailService);

}

@Test

@DisplayName("Verify method called with specific arguments")

public void testVerifySpecificArguments() {

// Arrange

when(mockUserRepository.findUserById(123)).thenReturn("John Doe");

// Act

String result = userService.getUserInfo(123);

// Assert

verify(mockUserRepository).findUserById(123); // Verify exact argument

assertEquals("John Doe", result);

System.out.println("✅ Test 1: Verified findUserById called with ID 123");

System.out.println(" Result: " + result);

}

@Test

@DisplayName("Verify method called specific number of times")

public void testVerifyNumberOfInvocations() {

// Arrange

when(mockUserRepository.findUserById(anyInt())).thenReturn("User");

List<Integer> userIds = Arrays.asList(1, 2, 3);

// Act

userService.processMultipleUsers(userIds);

// Assert

verify(mockUserRepository, times(3)).findUserById(anyInt());

verify(mockEmailService, times(3)).sendNotification(anyInt(), eq("Processing complete"));

System.out.println("✅ Test 2: Verified methods called 3 times each");

System.out.println(" Processed user IDs: " + userIds);

}

@Test

@DisplayName("Verify method never called")

public void testVerifyNeverCalled() {

// Arrange

when(mockUserRepository.saveUser(anyString(), anyInt(), anyString())).thenReturn(false);

// Act

boolean result = userService.createUser("John", 25, "john@email.com");

// Assert

verify(mockUserRepository).saveUser("John", 25, "john@email.com");

verify(mockEmailService, never()).sendWelcomeEmail(anyString(), anyString());

assertFalse(result);

System.out.println("✅ Test 3: Verified email service never called when save fails");

System.out.println(" User creation result: " + result);

}

@Test

@DisplayName("Verify method call order")

public void testVerifyCallOrder() {

// Arrange

when(mockUserRepository.findUserById(100)).thenReturn("Test User");

InOrder inOrder = inOrder(mockUserRepository, mockEmailService);

// Act

userService.removeUser(100);

// Assert - Verify calls happen in specific order

inOrder.verify(mockUserRepository).findUserById(100);

inOrder.verify(mockUserRepository).deleteUser(100);

inOrder.verify(mockEmailService).sendNotification(100, "Account deleted");

System.out.println("✅ Test 4: Verified method calls in correct order");

System.out.println(" Order: findUser → deleteUser → sendNotification");

}

@Test

@DisplayName("Verify using argument matchers")

public void testVerifyWithArgumentMatchers() {

// Arrange

when(mockUserRepository.findUsersByRole("admin")).thenReturn(Arrays.asList("Admin1", "Admin2"));

// Act

List<String> admins = userService.getAdminUsers();

// Assert

verify(mockUserRepository).findUsersByRole(eq("admin")); // Exact match

verify(mockUserRepository).findUsersByRole(contains("min")); // Contains

verify(mockUserRepository).findUsersByRole(startsWith("adm")); // Starts with

verify(mockUserRepository).findUsersByRole(matches("admin")); // Regex match

assertEquals(2, admins.size());

System.out.println("✅ Test 5: Verified with various argument matchers");

System.out.println(" Admin users found: " + admins);

}

@Test

@DisplayName("Verify with argument capture")

public void testVerifyWithArgumentCapture() {

// Arrange

when(mockUserRepository.isUserActive(500)).thenReturn(true);

ArgumentCaptor<String> emailCaptor = ArgumentCaptor.forClass(String.class);

ArgumentCaptor<String> subjectCaptor = ArgumentCaptor.forClass(String.class);

ArgumentCaptor<String> bodyCaptor = ArgumentCaptor.forClass(String.class);

// Act

userService.updateEmail(500, "newemail@example.com");

// Assert

verify(mockUserRepository).updateUserEmail(500, "newemail@example.com");

verify(mockEmailService).sendEmail(

emailCaptor.capture(),

subjectCaptor.capture(),

bodyCaptor.capture()

);

// Verify captured arguments

assertEquals("newemail@example.com", emailCaptor.getValue());

assertEquals("Email Updated", subjectCaptor.getValue());

assertEquals("Your email has been updated", bodyCaptor.getValue());

System.out.println("✅ Test 6: Verified with argument capture");

System.out.println(" Captured email: " + emailCaptor.getValue());

System.out.println(" Captured subject: " + subjectCaptor.getValue());

System.out.println(" Captured body: " + bodyCaptor.getValue());

}

@Test

@DisplayName("Verify complex interaction scenario")

public void testComplexInteractionVerification() {

// Arrange

when(mockUserRepository.saveUser("Alice", 30, "alice@email.com")).thenReturn(true);

when(mockUserRepository.getUserCount()).thenReturn(150);

when(mockUserRepository.findUsersByRole("admin")).thenReturn(Arrays.asList("Admin1", "Admin2"));

// Act

boolean userCreated = userService.createUser("Alice", 30, "alice@email.com");

String stats = userService.getSystemStats();

// Assert

assertTrue(userCreated);

assertEquals("Total Users: 150, Admins: 2", stats);

// Verify all interactions

verify(mockUserRepository).saveUser("Alice", 30, "alice@email.com");

verify(mockEmailService).sendWelcomeEmail("alice@email.com", "Alice");

verify(mockUserRepository).getUserCount();

verify(mockUserRepository).findUsersByRole("admin");

// Verify no other interactions

verifyNoMoreInteractions(mockUserRepository);

verifyNoMoreInteractions(mockEmailService);

System.out.println("✅ Test 7: Complex interaction verification completed");

System.out.println(" User created: " + userCreated);

System.out.println(" System stats: " + stats);

}

@Test

@DisplayName("Verify no interactions")

public void testVerifyNoInteractions() {

// Act - Don't call any service methods

// Assert

verifyNoInteractions(mockUserRepository);

verifyNoInteractions(mockEmailService);

System.out.println("✅ Test 8: Verified no interactions occurred");

}

@Test

@DisplayName("Verify with timeout")

public void testVerifyWithTimeout() {

// Arrange

when(mockUserRepository.findUserById(999)).thenReturn("Slow User");

// Act

String result = userService.getUserInfo(999);

// Assert - Verify call happens within timeout

verify(mockUserRepository, timeout(1000)).findUserById(999);

assertEquals("Slow User", result);

System.out.println("✅ Test 9: Verified method call within timeout");

System.out.println(" Result: " + result);

}

}

// 5. TestRunner.java (Execute all verification tests)

import org.junit.platform.launcher.Launcher;

import org.junit.platform.launcher.LauncherDiscoveryRequest;

import org.junit.platform.launcher.core.LauncherDiscoveryRequestBuilder;

import org.junit.platform.launcher.core.LauncherFactory;

import org.junit.platform.launcher.listeners.SummaryGeneratingListener;

import org.junit.platform.launcher.listeners.TestExecutionSummary;

import static org.junit.platform.engine.discovery.DiscoverySelectors.selectClass;

public class TestRunner {

public static void main(String[] args) {

System.out.println("🚀 Starting Mockito Verification Tests...\n");

LauncherDiscoveryRequest request = LauncherDiscoveryRequestBuilder.request()

.selectors(selectClass(UserServiceVerificationTest.class))

.build();

Launcher launcher = LauncherFactory.create();

SummaryGeneratingListener listener = new SummaryGeneratingListener();

launcher.registerTestExecutionListeners(listener);

launcher.execute(request);

TestExecutionSummary summary = listener.getSummary();

System.out.println("\n📊 Verification Test Results:");

System.out.println("════════════════════════════════");

System.out.println("Total tests: " + summary.getTestsFoundCount());

System.out.println("Tests succeeded: " + summary.getTestsSucceededCount());

System.out.println("Tests failed: " + summary.getTestsFailedCount());

System.out.println("Tests skipped: " + summary.getTestsSkippedCount());

System.out.println("Execution time: " + summary.getTotalTime().toMillis() + "ms");

if (summary.getTestsFailedCount() > 0) {

System.out.println("\n❌ Failed tests:");

summary.getFailures().forEach(failure ->

System.out.println("- " + failure.getTestIdentifier().getDisplayName() +

": " + failure.getException().getMessage())

);

} else {

System.out.println("\n✅ All verification tests passed successfully!");

System.out.println("\n🎯 Verification Techniques Demonstrated:");

System.out.println("• Specific argument verification");

System.out.println("• Number of invocations verification");

System.out.println("• Never called verification");

System.out.println("• Call order verification");

System.out.println("• Argument matchers");

System.out.println("• Argument capture");

System.out.println("• Complex interaction scenarios");

System.out.println("• No interactions verification");

System.out.println("• Timeout verification");

}

}

}

🚀 Starting Mockito Verification Tests...

✅ Test 1: Verified findUserById called with ID 123

Result: John Doe

✅ Test 2: Verified methods called 3 times each

Processed user IDs: [1, 2, 3]

✅ Test 3: Verified email service never called when save fails

User creation result: false

✅ Test 4: Verified method calls in correct order

Order: findUser → deleteUser → sendNotification

✅ Test 5: Verified with various argument matchers

Admin users found: [Admin1, Admin2]

✅ Test 6: Verified with argument capture

Captured email: newemail@example.com

Captured subject: Email Updated

Captured body: Your email has been updated

✅ Test 7: Complex interaction verification completed

User created: true

System stats: Total Users: 150, Admins: 2

✅ Test 8: Verified no interactions occurred

✅ Test 9: Verified method call within timeout

Result: Slow User

📊 Verification Test Results:

════════════════════════════════

Total tests: 9

Tests succeeded: 9

Tests failed: 0

Tests skipped: 0

Execution time: 156ms

✅ All verification tests passed successfully!

🎯 Verification Techniques Demonstrated:

• Specific argument verification

• Number of invocations verification

• Never called verification

• Call order verification

• Argument matchers

• Argument capture

• Complex interaction scenarios

• No interactions verification

• Timeout verification

=== Detailed Verification Examples ===

1. SPECIFIC ARGUMENTS:

verify(mockUserRepository).findUserById(123);

→ Verified exact method call with ID 123

2. NUMBER OF INVOCATIONS:

verify(mockUserRepository, times(3)).findUserById(anyInt());

→ Verified method called exactly 3 times

3. NEVER CALLED:

verify(mockEmailService, never()).sendWelcomeEmail(anyString(), anyString());

→ Verified method was never invoked

4. CALL ORDER:

InOrder inOrder = inOrder(mockUserRepository, mockEmailService);

inOrder.verify(mockUserRepository).findUserById(100);

inOrder.verify(mockUserRepository).deleteUser(100);

→ Verified methods called in specific sequence

5. ARGUMENT MATCHERS:

verify(mockUserRepository).findUsersByRole(eq("admin"));

verify(mockUserRepository).findUsersByRole(contains("min"));

verify(mockUserRepository).findUsersByRole(startsWith("adm"));

→ Verified using flexible argument matching

6. ARGUMENT CAPTURE:

ArgumentCaptor<String> emailCaptor = ArgumentCaptor.forClass(String.class);

verify(mockEmailService).sendEmail(emailCaptor.capture(), \*, \*);

assertEquals("newemail@example.com", emailCaptor.getValue());

→ Captured and verified actual argument values

7. NO MORE INTERACTIONS:

verifyNoMoreInteractions(mockUserRepository);

→ Ensured no unexpected method calls occurred

8. TIMEOUT VERIFICATION:

verify(mockUserRepository, timeout(1000)).findUserById(999);

→ Verified method call completed within time limit

=== Maven Test Output ===

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running UserServiceVerificationTest

[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.156 s

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] BUILD SUCCESS

[INFO] Total time: 3.421 s

[INFO] Finished at: 2024-01-15T10:30:45Z

=== Key Verification Patterns ===

✅ Basic Verification:

verify(mock).method(args);

✅ Times Verification:

verify(mock, times(n)).method(args);

verify(mock, atLeast(n)).method(args);

verify(mock, atMost(n)).method(args);

verify(mock, never()).method(args);

✅ Order Verification:

InOrder inOrder = inOrder(mock1, mock2);

inOrder.verify(mock1).method1();

inOrder.verify(mock2).method2();

✅ Argument Matching:

verify(mock).method(eq("exact"));

verify(mock).method(anyString());

verify(mock).method(contains("text"));

✅ Argument Capture:

ArgumentCaptor<Type> captor = ArgumentCaptor.forClass(Type.class);

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

assertEquals(expected, captor.getValue());

✅ Interaction Control:

verifyNoInteractions(mock);

verifyNoMoreInteractions(mock);

verify(mock, timeout(1000)).method();