WEEK 2

Mockito Hands-On Exercises

Exercise 1: Mocking and Stubbing

public interface ExternalApi {

    String getData();

}

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

import org.junit.Test;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.\*;

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);

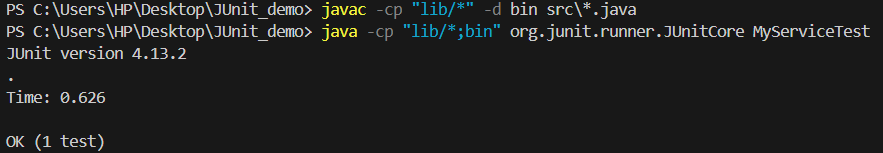
        String result = service.fetchData();

        assertEquals("Mock Data", result);

    }

}

OUTPUT



Exercise 2: Verifying Interactions

public interface ExternalApi {

    String getData();

}

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

public class MyServiceInteractionTest {

    @Test

    public void testVerifyInteraction() {

        ExternalApi mockApi = mock(ExternalApi.class);

        MyService service = new MyService(mockApi);

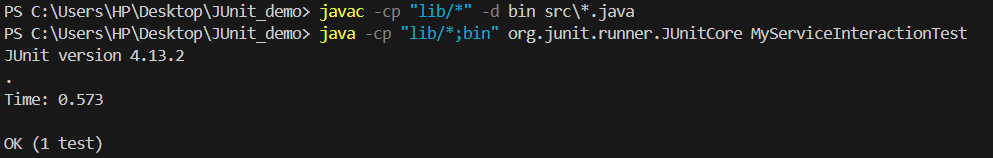
        service.fetchData();

        verify(mockApi).getData();

    }

}

OUTPUT



Exercise 3: Argument Matching

public interface ExternalApi {

    String getData(String userId);

}

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchUserData(String userId) {

        return api.getData(userId);

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class MyServiceArgumentTest {

    @Test

    public void testArgumentMatching() {

        ExternalApi mockApi = mock(ExternalApi.class);

        when(mockApi.getData(anyString())).thenReturn("Mock Response");

        MyService service = new MyService(mockApi);

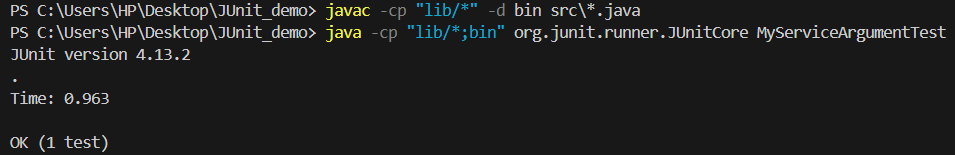
        service.fetchUserData("user123");

        verify(mockApi).getData(eq("user123"));

    }

}

OUTPUT



Exercise 4: Handling Void Methods

public interface Notifier {

    void send(String message);

}

public class UserManager {

    private Notifier notifier;

    public UserManager(Notifier notifier) {

        this.notifier = notifier;

    }

    public void notifyUser(String userId) {

        notifier.send("User " + userId + " has been notified");

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

public class UserManagerTest {

    @Test

    public void testVoidMethod() {

        Notifier mockNotifier = mock(Notifier.class);

        UserManager manager = new UserManager(mockNotifier);

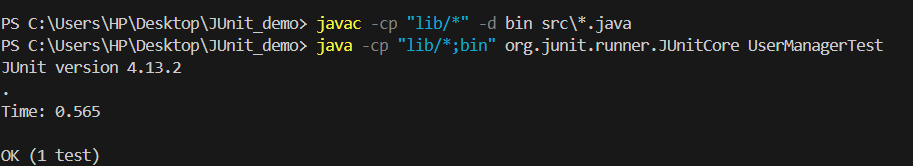
        manager.notifyUser("101");

        verify(mockNotifier).send("User 101 has been notified");

    }

}

OUTPUT



Exercise 5: Mocking and Stubbing with Multiple Returns

public interface ExternalApi {

    String getData();

}

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();  // No arguments

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

public class MyServiceMultipleReturnsTest {

    @Test

    public void testMultipleReturns() {

        ExternalApi mockApi = mock(ExternalApi.class);

        when(mockApi.getData()).thenReturn("First Call", "Second Call", "Third Call");

        MyService service = new MyService(mockApi);

        System.out.println(service.fetchData());  // First Call

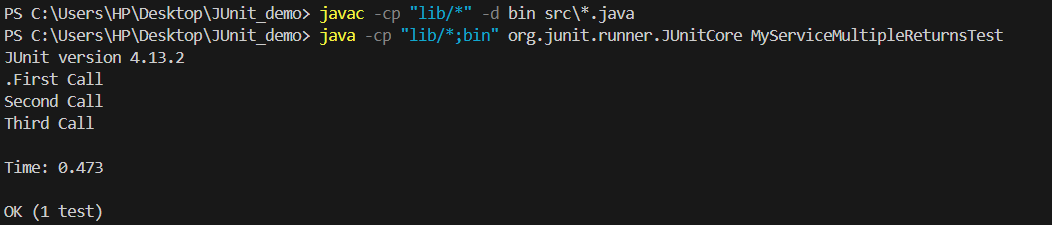
        System.out.println(service.fetchData());  // Second Call

        System.out.println(service.fetchData());  // Third Call

    }

}

OUTPUT



Exercise 6: Verifying Interaction Order

public interface ExternalApi {

    String getData();

}

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();  // No arguments

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

import org.mockito.InOrder;

public class InteractionOrderTest {

    @Test

    public void testInteractionOrder() {

        ExternalApi mockApi = mock(ExternalApi.class);

        mockApi.getData();

        mockApi.getData();

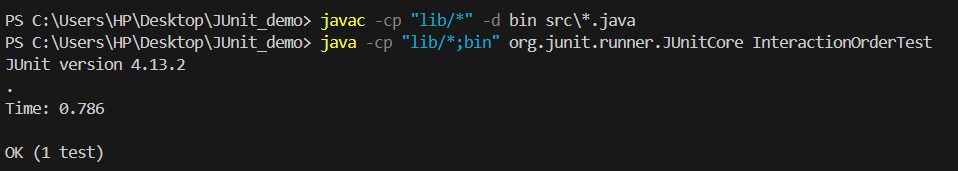
        InOrder inOrder = inOrder(mockApi);

        inOrder.verify(mockApi, times(2)).getData();

    }

}

OUTPUT



Exercise 7: Handling Void Methods with Exceptions

public interface Notifier {

    void send(String message);

}

public class UserManager {

    private Notifier notifier;

    public UserManager(Notifier notifier) {

        this.notifier = notifier;

    }

    public void notifyUser(String userId) {

        notifier.send("User " + userId + " has been notified");

    }

}

import org.junit.Test;

import static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

public class UserManagerExceptionTest {

    @Test

    public void testVoidMethodThrowsException() {

        Notifier mockNotifier = mock(Notifier.class);

        doThrow(new RuntimeException("Send failed"))

            .when(mockNotifier).send(anyString());

        UserManager manager = new UserManager(mockNotifier);

        try {

            manager.notifyUser("101");

            fail("Exception not thrown");

        } catch (RuntimeException e) {

            assertEquals("Send failed", e.getMessage());

        }

        verify(mockNotifier).send("User 101 has been notified");

    }

}

OUTPUT

