**WEEK 2: Mockito Hands-On Exercises**

**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.

**Code:**

**MockingStubbingExample.java**

import static org.mockito.Mockito.\*;

public class MockingStubbingExample {

    interface ExternalApi {

        String getData();

        int getCount();

    }

    static class MyService {

        private ExternalApi api;

        public MyService(ExternalApi api) {

            this.api = api;

        }

        public void printData() {

            System.out.println("Data: " + api.getData());

            System.out.println("Count: " + api.getCount());

        }

    }

    public static void main(String[] args) {

        ExternalApi mockApi = mock(ExternalApi.class);

        when(mockApi.getData()).thenReturn("Hello World");

        when(mockApi.getCount()).thenReturn(42);

        MyService service = new MyService(mockApi);

        service.printData();

        verify(mockApi).getData();

        verify(mockApi).getCount();

    }

}

**Output:**



**Exercise 2:** Verifying Interactions

**Scenario:** You need to ensure that a method is called with specific arguments.

**Code:**

**VerifyingInteractionsExample.java**

import static org.mockito.Mockito.\*;

public class VerifyingInteractionsExample {

    interface MyService {

        void sendMessage(String message, int times);

    }

    public static void main(String[] args) {

        MyService mockService = mock(MyService.class);

        mockService.sendMessage("Hello Mockito!", 3);

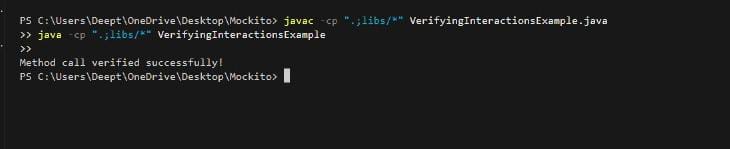
        verify(mockService).sendMessage("Hello Mockito!", 3);

        System.out.println("Method call verified successfully!");

    }

}

**Output:**



**Exercise 3:** Argument Matching

**Scenario:** You need to verify that a method is called with specific arguments.

**Code:**

**ArgumentMatchingExample.java**

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class ArgumentMatchingExample {

interface MyService {

void process(String name, int age);

}

public static void main(String[] args) {

MyService mockService = mock(MyService.class);

mockService.process("Deepthi", 30);

verify(mockService).process("Deepthi", 30);

verify(mockService).process(anyString(), anyInt());

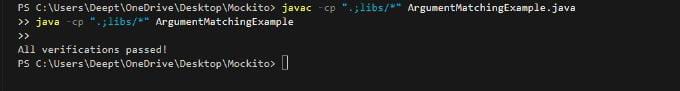
verify(mockService).process(startsWith("D"), anyInt());

System.out.println("All verifications passed!");

}

}

**Output:**



**Exercise 4:** Handling Void Methods

**Scenario:** You need to test a void method that performs some action.

**Code:**

**VoidMethodExample.java**

import static org.mockito.Mockito.\*;

public class VoidMethodExample {

    interface MyService {

        void performAction(String input);

    }

    public static void main(String[] args) {

        MyService mockService = mock(MyService.class);

        doNothing().when(mockService).performAction("TestInput");

        mockService.performAction("TestInput");

        verify(mockService).performAction("TestInput");

        System.out.println("Void method verified successfully!");

    }

}

**Output:**



**Exercise 5:** Mocking and Stubbing with Multiple Returns

**Scenario:** You need to test a service that depends on an external API with multiple return values.

**Code:**

**MultipleReturnsExample.java**

import static org.mockito.Mockito.\*;

public class MultipleReturnsExample {

    interface ExternalApi {

        String fetchData();

    }

    static class MyService {

        private ExternalApi api;

        public MyService(ExternalApi api) {

            this.api = api;

        }

        public void printData() {

            System.out.println("Data: " + api.fetchData());

        }

    }

    public static void main(String[] args) {

        ExternalApi mockApi = mock(ExternalApi.class);

        when(mockApi.fetchData())

            .thenReturn("First call result")

            .thenReturn("Second call result")

            .thenReturn("Third call result");

        MyService service = new MyService(mockApi);

        service.printData();

        service.printData();

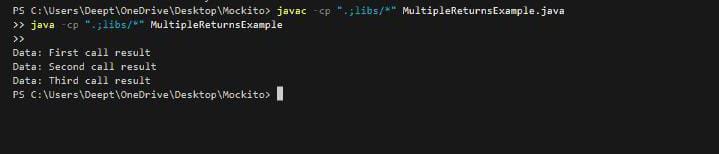
        service.printData();

        verify(mockApi, times(3)).fetchData();

    }

}

**Output:**



**Exercise 6:** Verifying Interaction Order

**Scenario:** You need to ensure that methods are called in a specific order.

**Code:**

**InteractionOrderExample.java**

import static org.mockito.Mockito.\*;

import org.mockito.InOrder;

public class InteractionOrderExample {

    interface MyService {

        void firstMethod();

        void secondMethod();

        void thirdMethod();

    }

    public static void main(String[] args) {

        MyService mockService = mock(MyService.class);

        mockService.firstMethod();

        mockService.secondMethod();

        mockService.thirdMethod();

        InOrder inOrder = inOrder(mockService);

        inOrder.verify(mockService).firstMethod();

        inOrder.verify(mockService).secondMethod();

        inOrder.verify(mockService).thirdMethod();

        System.out.println("Methods were called in the correct order!");

    }

}

**Output:**



**Exercise 7:** Handling Void Methods with Exceptions

**Scenario:** You need to test a void method that throws an exception.

**Code:**

**VoidMethodExceptionExample.java**

import static org.mockito.Mockito.\*;

public class VoidMethodExceptionExample {

    interface MyService {

        void riskyOperation() throws Exception;

    }

    public static void main(String[] args) throws Exception {

        MyService mockService = mock(MyService.class);

        doThrow(new RuntimeException("Something went wrong"))

            .when(mockService)

            .riskyOperation();

        try {

            mockService.riskyOperation();

        } catch (RuntimeException e) {

            System.out.println("Caught exception: " + e.getMessage());

        }

        verify(mockService).riskyOperation();

        System.out.println("Void method with exception verified!");

    }

}

**Output:**

