Mockito Excercises:

**Exercise 3: Argument Matching**

UserRepository.java

package com.example;

public interface UserRepository {

    void saveUser(String username);

}

UserService.java

package com.example;

public class UserService {

    private UserRepository repository;

    public UserService(UserRepository repository) {

        this.repository = repository;

    }

    public void registerUser(String username) {

        repository.saveUser(username);

    }

}

UserServiceTest.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class UserServiceTest {

    @Test

    public void testArgumentMatching() {

        UserRepository mockRepo = mock(UserRepository.class);

        UserService service = new UserService(mockRepo);

        service.registerUser("Alice12");

        verify(mockRepo).saveUser(argThat(name -> name.startsWith("Alice")));

    }

}

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>demo</artifactId>

    <version>1.0-SNAPSHOT</version>

    <properties>

        <maven.compiler.source>21</maven.compiler.source>

        <maven.compiler.target>21</maven.compiler.target>

    </properties>

<dependencies>

    <dependency>

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

        <artifactId>junit-jupiter</artifactId>

        <version>5.10.0</version>

        <scope>test</scope>

    </dependency>

    <dependency>

        <groupId>org.mockito</groupId>

        <artifactId>mockito-core</artifactId>

        <version>5.12.0</version>

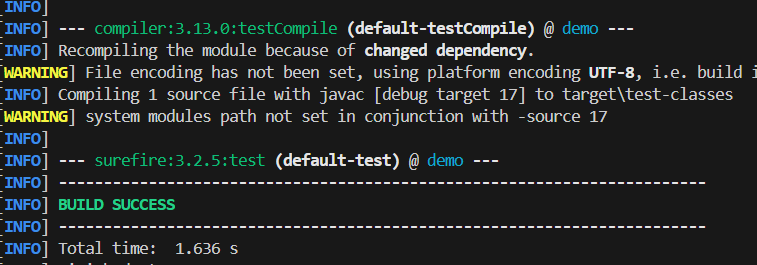
        <scope>test</scope>

    </dependency>

</dependencies>

</project>

**Output:**

****

**Exercise 4: Handling Void Methods**

**Notifier.java**

package com.example;

public interface Notifier {

    void sendNotification(String message);

}

**NotificationService.java**

package com.example;

public class NotificationService {

    private final Notifier notifier;

    public NotificationService(Notifier notifier) {

        this.notifier = notifier;

    }

    public void process() {

        notifier.sendNotification("Hello");

    }

}

**NotificationServiceTest.java**

package com.example;

import org.junit.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

public class NotificationServiceTest {

    @Test

    public void testSendNotificationCalled() {

        Notifier mockNotifier = mock(Notifier.class);

        NotificationService service = new NotificationService(mockNotifier);

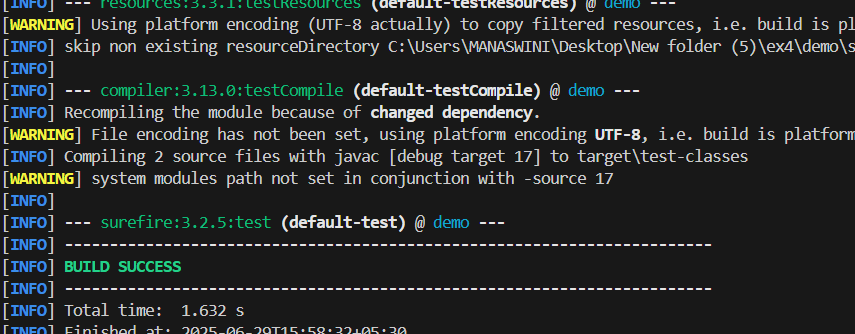
        service.process();

        verify(mockNotifier).sendNotification("Hello from service!");

    }

}

**Output:**

****

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

**ExternalApi.java**

package com.example;

public interface ExternalApi {

    String getData();

}

**MyService.java**

package com.example;

public class MyService {

    private final ExternalApi externalApi;

    public MyService(ExternalApi externalApi) {

        this.externalApi = externalApi;

    }

    public String[] fetchMultipleData() {

        String a = externalApi.getData();

        String b = externalApi.getData();

        return new String[]{first, second};

    }

**}**

**MyServiceTest.java**

package com.example;

import org.junit.Test;

import org.mockito.Mockito;

import static org.junit.Assert.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

    @Test

    public void testMultipleReturns() {

        ExternalApi mockApi = Mockito.mock(ExternalApi.class);

        when(mockApi.getData())

            .thenReturn("First value")

            .thenReturn("Second value");

        MyService service = new MyService(mockApi);

        String[] result = service.fetchMultipleData();

        assertEquals("First value", result[0]);

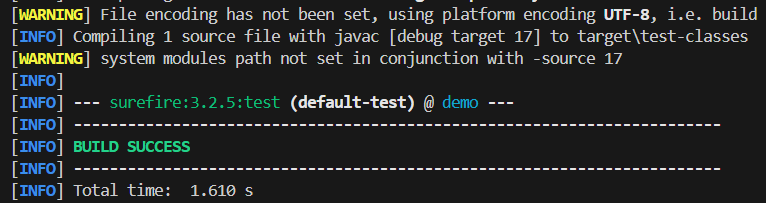
        assertEquals("Second value", result[1]);

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

    }

}

**Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercise 6: Verifying Interaction Order**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

    void initialize();

    void fetchData();

    void shutdown();

}

**MyService.java**

package com.example;

public class MyService {

    private final ExternalApi externalApi;

    public MyService(ExternalApi externalApi) {

        this.externalApi = externalApi;

    }

    public void process() {

        externalApi.initialize();

        externalApi.fetchData();

        externalApi.shutdown();

    }

}

**MyServiceTest.java**

package com.example;

import org.junit.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

    @Test

    public void testInteractionOrder() {

        ExternalApi mockApi = mock(ExternalApi.class);

        MyService service = new MyService(mockApi);

        service.process();

        InOrder inOrder = inOrder(mockApi);

        inOrder.verify(mockApi).initialize();

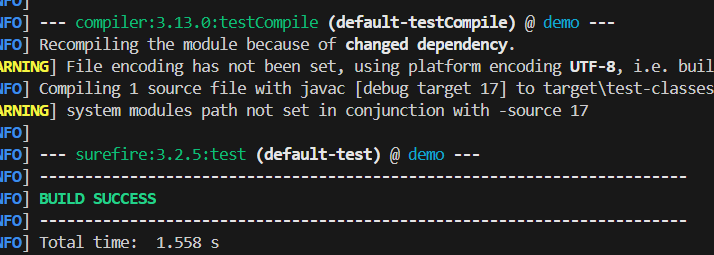
        inOrder.verify(mockApi).fetchData();

        inOrder.verify(mockApi).shutdown();

    }

}

**Output:**

****

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

**ExternalApi.java**

package com.example;

public interface ExternalApi {

    void Operation();

}

**MyService.java**

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void perform() {

externalApi.Operation();

}

}

**MyServiceTest.java**

package com.example;

import org.junit.Test;

import static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

public class MyServiceTest {

    @Test(expected = RuntimeException.class)

    public void testVoidMethodThrowsException() {

        ExternalApi mockApi = mock(ExternalApi.class);

        doThrow(new RuntimeException("Failure")).when(mockApi).Operation();

        MyService service = new MyService(mockApi);

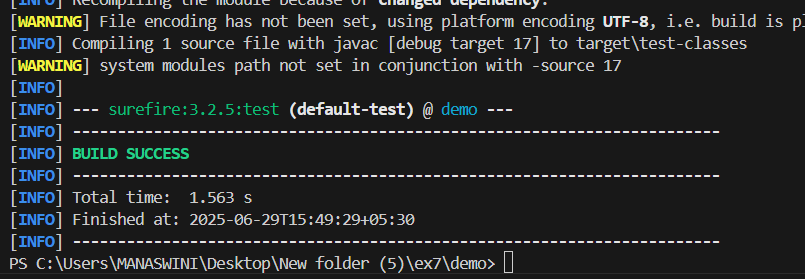
        service.perform();

        verify(mockApi).Operation();

    }

}

**Output:**

****