Mockito exercises

Exercise 1: Mocking and Stubbing

Scenario:

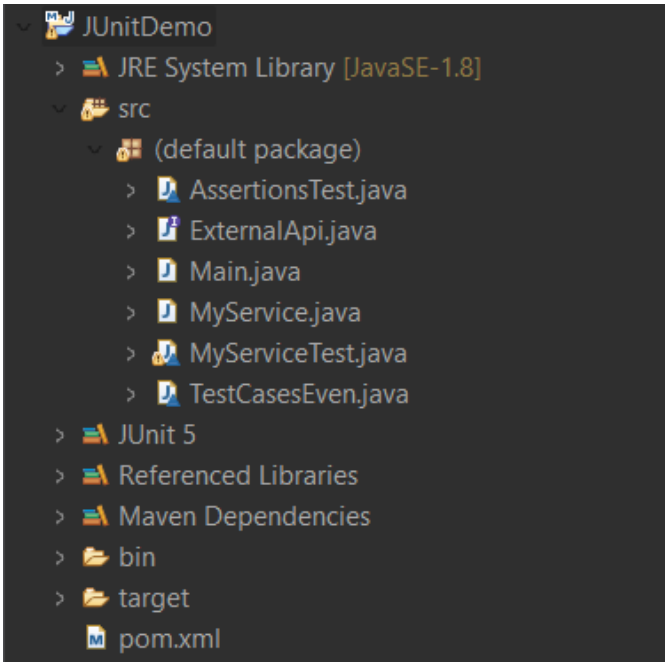
You need to test a service that depends on an external API. Use Mockito to mock the
external API and stub its methods.

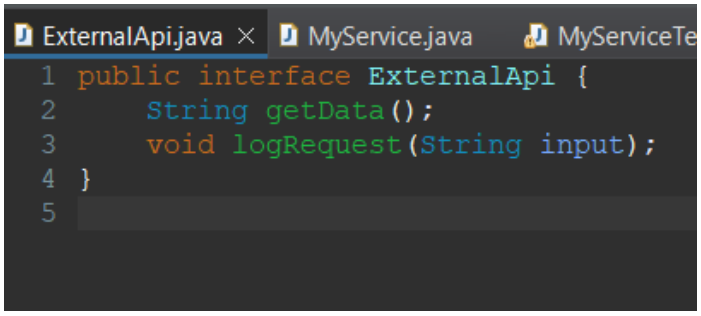
Steps:

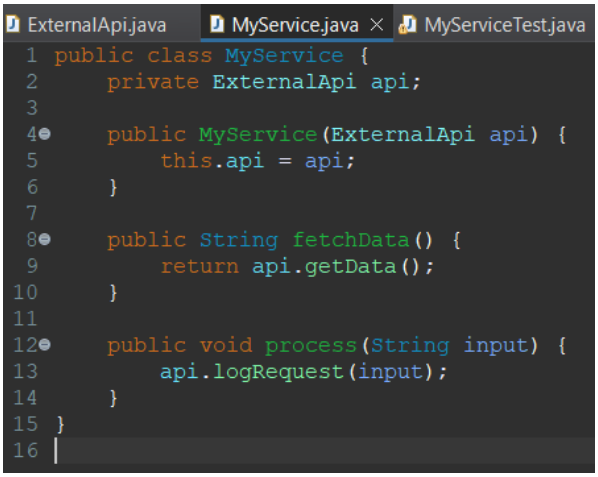
1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

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);
}
}
```







Main.java | TestCasesEven.java | AssertionsTest.java | ExternalApi.java | MyService.java | MyServiceTest.java | JUnitDemo/pom.xml

```java
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class MyServiceTest {

    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");

        MyService service = new MyService(mockApi);
        String result = service.fetchData();

        assertEquals("Mock Data", result);
    }

}
```

Console | Debug | JUnit

Finished after 2.034 seconds

Runs: 1/1 Errors: 0 Failures: 0

MyServiceTest [Runner: JUnit 5] (1.615 s)

Failure Trace

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.

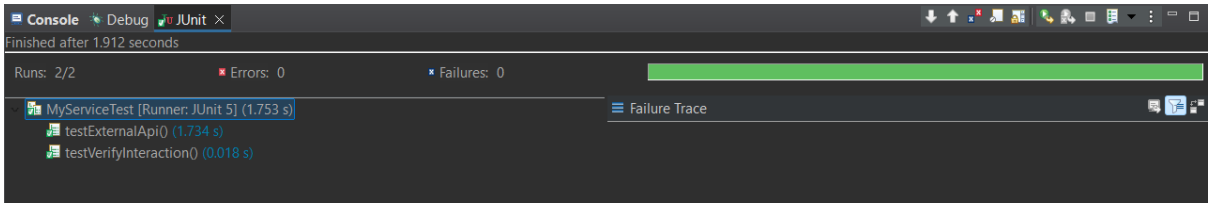
Solution Code:

```
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();
}
}
```

```java
@Test
public void testVerifyInteraction() {
    ExternalApi mockApi = mock(ExternalApi.class);
    MyService service = new MyService(mockApi);

    service.fetchData();

    verify(mockApi).getData(); // Verifies getData() was called
}
```



Exercise 3: Argument Matching

Scenario:

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

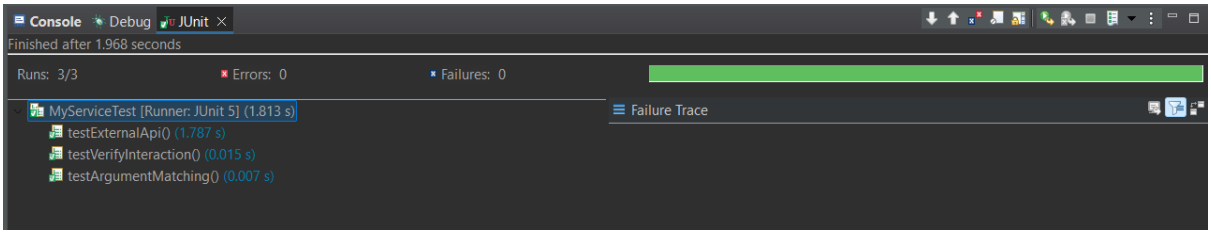
Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Use argument matchers to verify the interaction.

```java
@Test
public void testArgumentMatching() {
    ExternalApi mockApi = mock(ExternalApi.class);
    MyService service = new MyService(mockApi);

    service.process("Request123");

    verify(mockApi).logRequest(eq("Request123"));
}
```



Exercise 4: Handling Void Methods

Scenario:

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

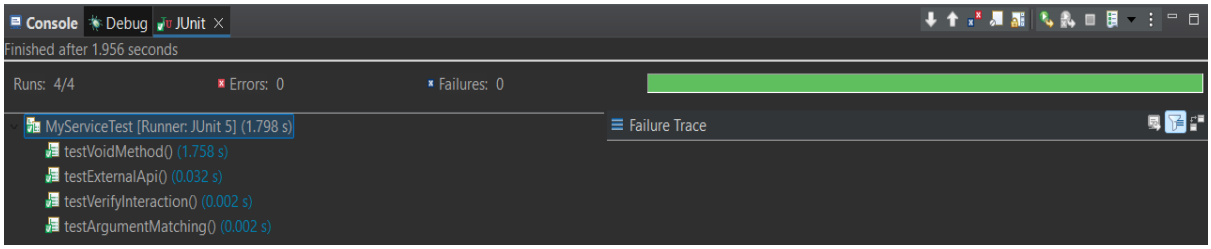
Steps:

1. Create a mock object.
2. Stub the void method.
3. Verify the interaction.

```
@Test
public void testVoidMethod() {
    ExternalApi mockApi = mock(ExternalApi.class);
    doNothing().when(mockApi).logRequest(anyString());

    MyService service = new MyService(mockApi);
    service.process("Hello");

    verify(mockApi).logRequest("Hello");
}
```



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.

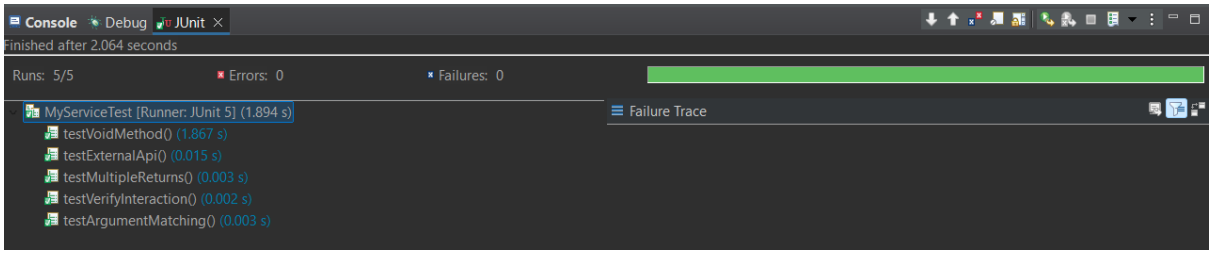
Steps:

1. Create a mock object for the external API.
2. Stub the methods to return different values on consecutive calls.
3. Write a test case that uses the mock object.

```
@Test
public void testMultipleReturns() {
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.getData())
        .thenReturn("First Call")
        .thenReturn("Second Call");

    MyService service = new MyService(mockApi);

    assertEquals("First Call", service.fetchData());
    assertEquals("Second Call", service.fetchData());
}
```



Exercise 6: Verifying Interaction Order

Scenario:

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

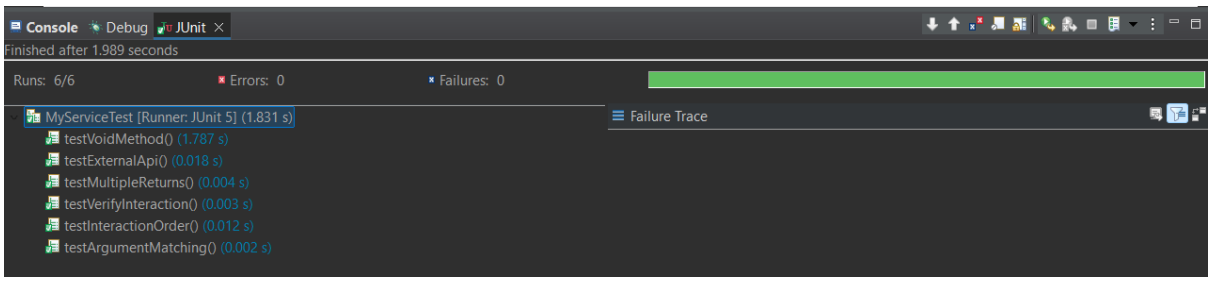
Steps:

1. Create a mock object.
2. Call the methods in a specific order.
3. Verify the interaction order.

```java
@Test
public void testInteractionOrder() {
    ExternalApi mockApi = mock(ExternalApi.class);
    MyService service = new MyService(mockApi);

    service.process("Step1");
    service.fetchData();

    InOrder order = inOrder(mockApi);
    order.verify(mockApi).logRequest("Step1");
    order.verify(mockApi).getData();
}
```



Exercise 7: Handling Void Methods with Exceptions

Scenario:

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

Steps:

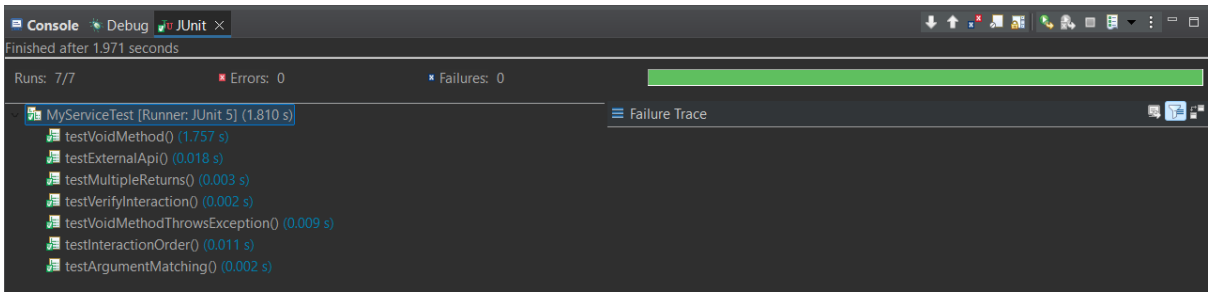
1. Create a mock object.
2. Stub the void method to throw an exception.
3. Verify the interaction.

```java
@Test
public void testVoidMethodThrowsException() {
    ExternalApi mockApi = mock(ExternalApi.class);
    doThrow(new RuntimeException("Error")).when(mockApi).logRequest("Crash");

    MyService service = new MyService(mockApi);

    assertThrows(RuntimeException.class, () -> service.process("Crash"));

    verify(mockApi).logRequest("Crash");
}
```



Submitted By:

Name : Lingaraj Nayak

Superset ID : 6387607