JUnit Testing 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:

```java
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

```xml
<dependency>
   <groupId>org.mockito</groupId>
   <artifactId>mockito-core</artifactId>
   <version>4.11.0</version>
```

```
    <scope>test</scope>
</dependency>
```

ExternalApi.java

```
package com.example;

public interface ExternalApi {
    String getData();
}
```

MyService.java

```
package com.example;

public class MyService {
    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData();
    }
}
```

MyServiceTest.java

```
package com.example;

import org.junit.Test;
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        // Step 1: Create mock
        ExternalApi mockApi = mock(ExternalApi.class);

        // Step 2: Stub the method
        when(mockApi.getData()).thenReturn("Mock Data");

        // Step 3: Use mock in service
        MyService service = new MyService(mockApi);

        // Step 4: Assert the expected result
        String result = service.fetchData();
        assertEquals("Mock Data", result);
    }
}
```

Explanation of Key Concepts Used

In this exercise, Mockito was used to isolate the service class from its external dependency. The mock() method from Mockito creates a fake implementation of the ExternalApi interface. This means the test doesn't need a real API server or actual network calls — the behavior is simulated.

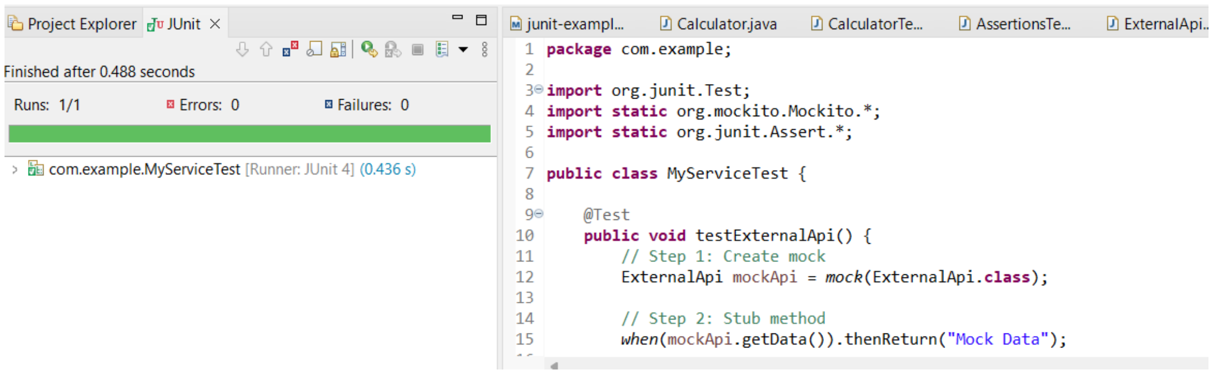
The when(...).thenReturn(...) syntax is used to stub a method. This allows us to define what the mock should return when its method is called. In this case, when getData() is called on the mocked API, it returns "Mock Data".

We use constructor-based dependency injection to pass the mocked API into the MyService class. This ensures that the service uses the mock rather than a real implementation during the test.

Finally, we use assertEquals() from JUnit to verify that the service method returns the expected value, which proves that the mock was correctly injected and used.

This combination of mocking, stubbing, and assertion makes unit testing more effective by ensuring tests focus only on the logic inside the service, without depending on external systems**.**

OUTPUT:



```
Finished after 0.488 seconds

Runs: 1/1          Errors: 0          Failures: 0

com.example.MyServiceTest [Runner: JUnit 4] (0.436 s)
```

```java
package com.example;

import org.junit.Test;
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        // Step 1: Create mock
        ExternalApi mockApi = mock(ExternalApi.class);

        // Step 2: Stub method
        when(mockApi.getData()).thenReturn("Mock Data");
```