

Week 2 - 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);
    }
}

ExternalApi.java
package com;
```

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

MyService.java

```
package com;
```

```
public class MyService {  
    private ExternalApi api;  
  
    public MyService(ExternalApi api) {  
        this.api = api;  
    }  
  
    public String fetchData() {  
        return api.getData(); // real call will be mocked  
    }  
}
```

MyServiceTest.java

```
package com;
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.mock;  
import static org.mockito.Mockito.when;
```

```
public class MyServiceTest {  
  
    @Test  
    public void testExternalApi() {
```

```
// Step 1: Create a mock
```

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

```
// Step 2: Stub method
```

```
when(mockApi.getData()).thenReturn("Mock Data");
```

```
// Step 3: Inject mock into service
```

```
MyService service = new MyService(mockApi);
```

```
// Step 4: Assert expected behavior
```

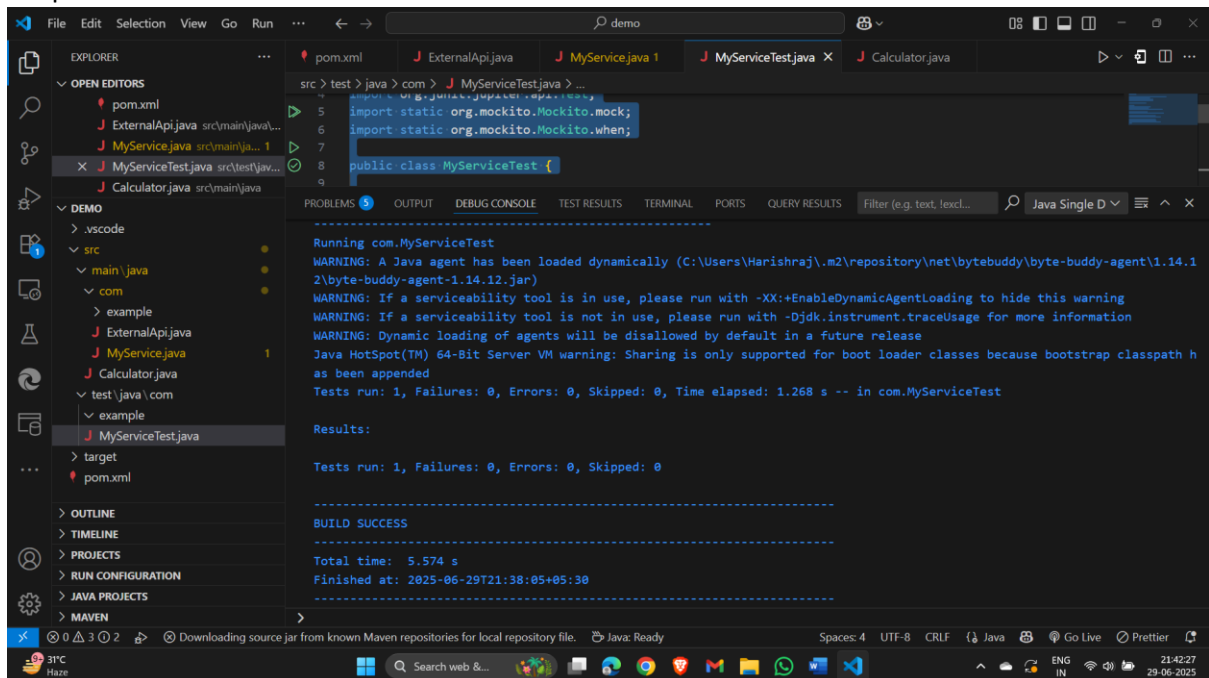
```
String result = service.fetchData();
```

```
assertEquals("Mock Data", result);
```

```
}
```

```
}
```

Output:



```
src > test > java > com > J MyServiceTest.java > ...
5 import static org.mockito.Mockito.mock;
6 import static org.mockito.Mockito.when;
7
8 public class MyServiceTest {
9
Running com.MyServiceTest
WARNING: A Java agent has been loaded dynamically (C:\Users\Harishraj\.m2\repository\net\bytebuddy\byte-buddy-agent\1.14.12\byte-buddy-agent-1.14.12.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.268 s -- in com.MyServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

-----
BUILD SUCCESS
-----
Total time: 5.574 s
Finished at: 2025-06-29T21:38:05+05:30
-----
```

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();
```

```
    }
```

```
}
```

ExternalApi.java

```
public interface ExternalApi {
```

```
    String getData();
```

```
}
```

MyService.java

```
public class MyService {
```

```
private ExternalApi api;
```

```
public MyService(ExternalApi api) {
```

```
    this.api = api;
```

```
}
```

```
public String fetchData() {
```

```
    return api.getData();
```

```
}
```

```
}
```

MyServiceTest.java

```
public class MyServiceTest {
```

```
    @Test
```

```
    public void testVerifyInteraction() {
```

```
        // Step 1: Create mock
```

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

```
        // Step 2: Use the mock in the service
```

```
        MyService service = new MyService(mockApi);
```

```
        service.fetchData(); // This should call mockApi.getData()
```

```
        // Step 3: Verify interaction
```

```
        verify(mockApi).getData(); // This passes only if getData() was called
```

```
    }
```

```
}
```

Output:

The screenshot displays an IDE interface with the following components:

- Explorer:** Shows the project structure. The 'test' directory is expanded, showing 'MyServiceTest.java'.
- Editor:** Displays the code for 'MyServiceTest.java'. The code includes imports for JUnit and Mockito, and a test method 'testVerifyInteraction()'.

```
src > test > java > J MyServiceTest.java > ...  
1  
2  
3 import org.junit.jupiter.api.Test;  
4 import static org.mockito.Mockito.mock;
```
- Output:** Shows the test results. The 'Test Runner for Java' section lists several test runs, all of which passed.
 - testVerifyInteraction() \$symbol...
 - 7 older results
 - Test run at 6/29/2025, 9:37:58 PM
 - Test run at 6/29/2025, 9:08:31 PM
 - Test run at 6/29/2025, 9:05:06 PM
 - Test run at 6/29/2025, 8:58:35 PM
 - Test run at 6/29/2025, 8:52:31 PM
 - Test run at 6/29/2025, 8:51:38 PM
 - Test run at 6/29/2025, 8:50:09 PM