Week2

TDD Using Junit5 and Mockito_HandsOn

Exercise 1: Setting Up JUnit

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your

pom.xml:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

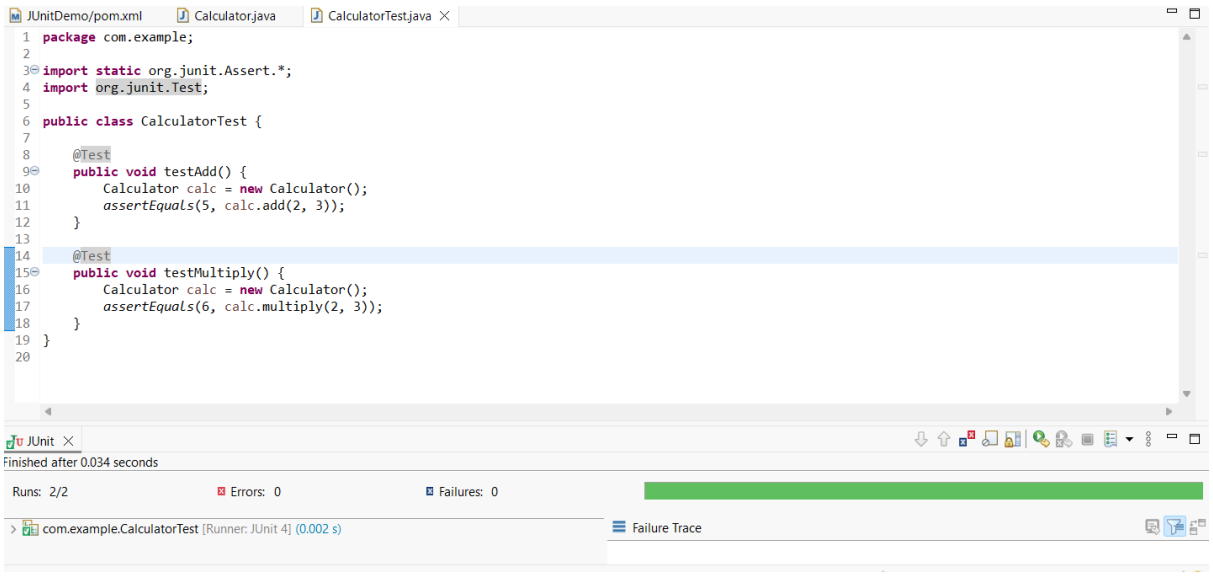
3. Create a new test class in your project.

CODE :

```
package com.example;
import static org.junit.Assert.*;
import org.junit.Test;
public class CalculatorTest {
    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
```

```
    }
    @Test
    public void testMultiply() {
        Calculator calc = new Calculator();
        assertEquals(6, calc.multiply(2, 3));
    }
}
```

OUTPUT :



Exercise 3: Assertions in JUnit

Scenario:

You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

Solution Code:

```
public class AssertionsTest {
@Test
public void testAssertions() {
```

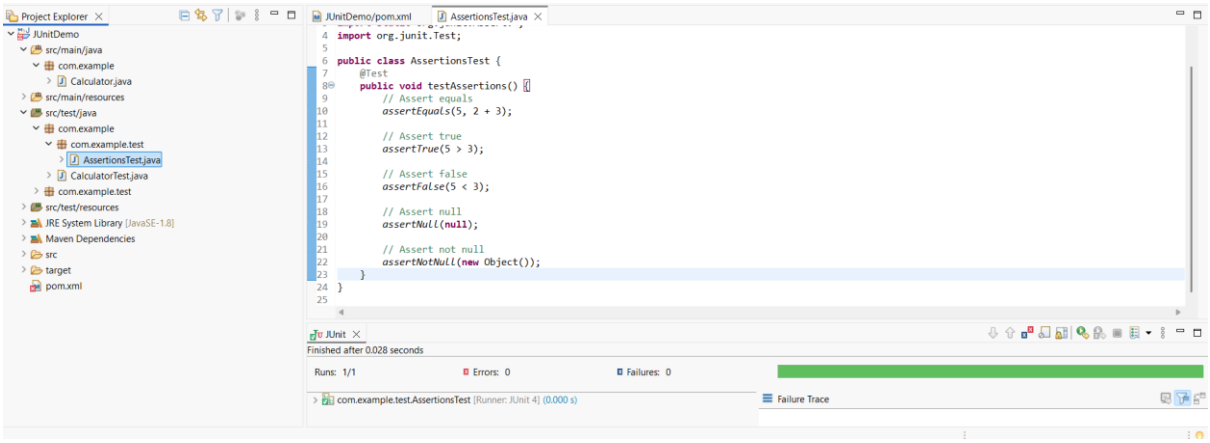
```
// Assert equals
assertEquals(5, 2 + 3);
// Assert true
assertTrue(5 > 3);
// Assert false
assertFalse(5 < 3);
// Assert null
assertNull(null);
// Assert not null
assertNotNull(new Object());
}
}
```

CODE :

```
package com.example.test;
import static org.junit.Assert.*;
import org.junit.Test;
public class AssertionsTest {
    @Test
    public void testAssertions() {
        // Assert equals
        assertEquals(5, 2 + 3);
        // Assert true
        assertTrue(5 > 3);
        // Assert false
        assertFalse(5 < 3);
        // Assert null
```

```
        assertNull(null);
        // Assert not null
        assertNotNull(new Object());
    }
}
```

OUTPUT :



Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup

and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

CODE :

Calculator.java

```
package com.example;

public class Calculator {
    public int add(int a, int b) {
```

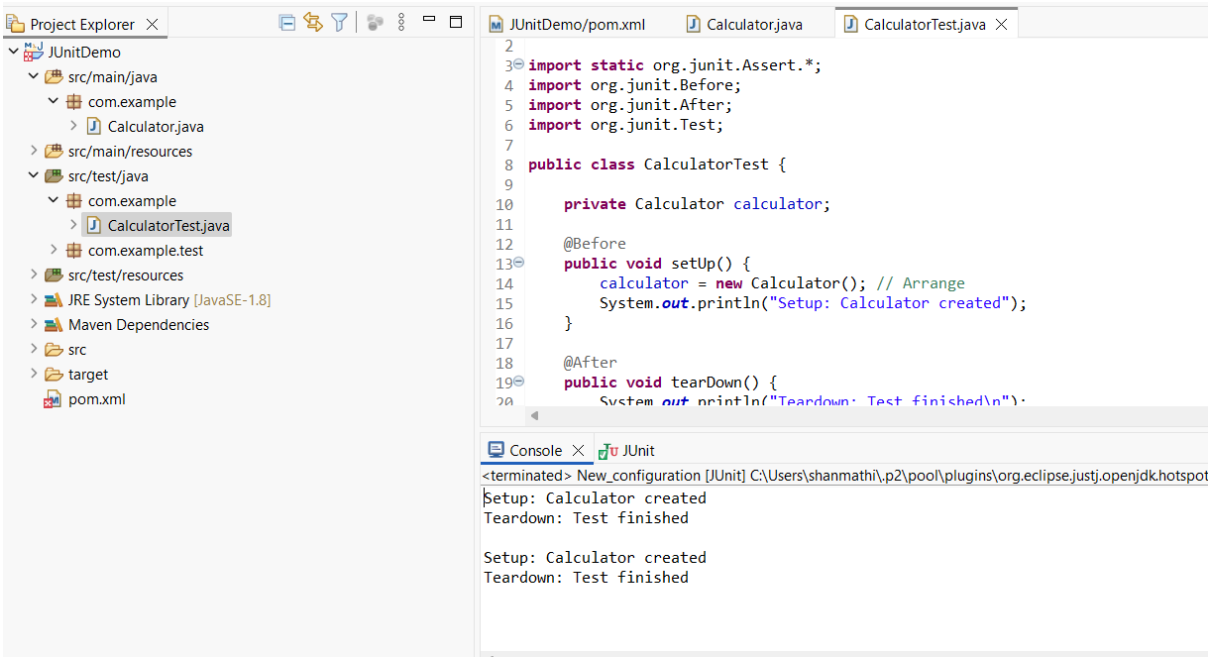
```
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}
```

CalculatorTest.java

```
package com.example;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
public class CalculatorTest {
    private Calculator calculator;
    @Before
    public void setUp() {
        calculator = new Calculator(); // Arrange
        System.out.println("Setup: Calculator created");
    }
    @After
    public void tearDown() {
        System.out.println("Teardown: Test finished\n");
    }
    @Test
    public void testAdd() {
```

```
        // Act
        int result = calculator.add(2, 3);
        // Assert
        assertEquals(5, result);
    }
    @Test
    public void testMultiply() {
        // Act
        int result = calculator.multiply(3, 4);
        // Assert
        assertEquals(12, result);
    }
}
```

OUTPUT :



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);

}

}
```

CODE :

ExternalApi.java

```
package com.example.demo;
public interface ExternalApi {
   String getData();
}
```

MyService.java

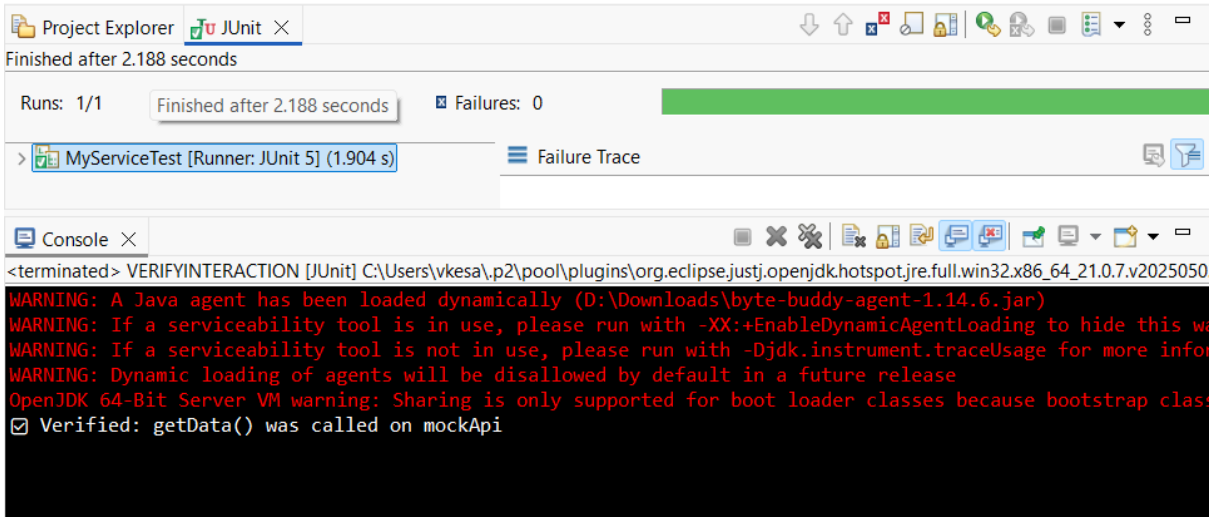
```
package com.example.demo
public class MyService {
   private ExternalApi externalApi;
   public MyService(ExternalApi externalApi) {
      this.externalApi = externalApi;
   }
   public String fetchData() {
      return externalApi.getData();
   }
}
```

MyServiceTest.java

```
package com.example.demo;
import static org.junit.jupiter.api.Assertions.assertEquals;
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);
        System.out.println("Test Passed: " + result);
    }
}
```

OUTPUT :



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();
}
}
```

CODE :

ExternalApi.java

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

MyService.java

```
package VERIFYINTERACTION;
public class MyService {
   private ExternalApi externalApi;
```

```
    public MyService(ExternalApi externalApi) {
        this.externalApi = externalApi;
    }

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

MyServiceTest.java

```
package VERIFYINTERACTION;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
public class MyServiceTest {
    @Test
    public void testVerifyInteraction() {
        // Step 1: Create a mock object
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);

        // Step 2: Call the method using MyService
        MyService service = new MyService(mockApi);
        service.fetchData();

        // Step 3: Verify that getData() was called
        verify(mockApi).getData();
```

```
        System.out.println("Verified: getData() was called on mockApi");
    }
}
```

OUTPUT :