TDD using Junit5 and Mockito – [Junit5 Basic Testing Exercise]

Exercise 1: Setting Up JUnit

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

Steps:

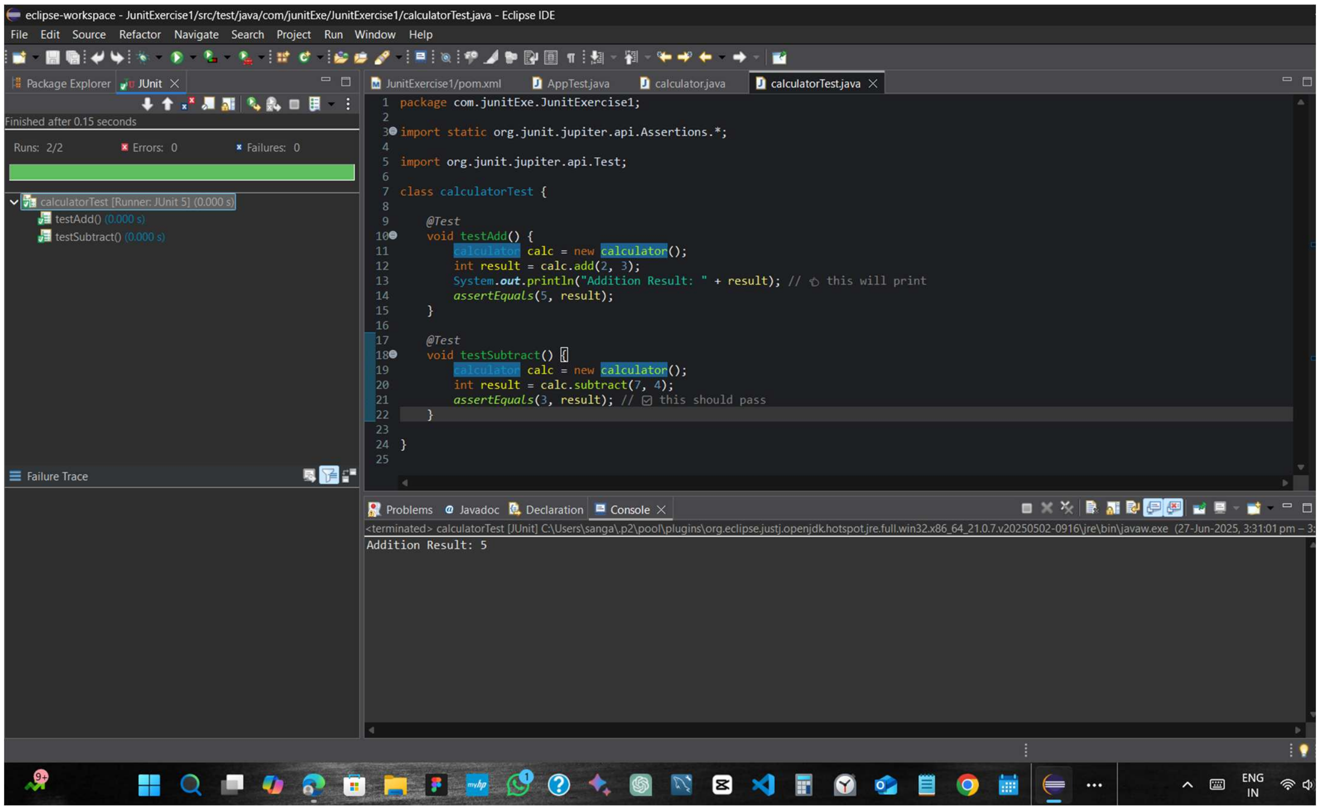
1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

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.

Result:


eclipse-workspace - JunitExercise1/src/test/java/com/junitExe/JunitExercise1/calculatorTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
JunitExercise1/pom.xml
AppTest.java
calculator.java
calculatorTest.java
Problems
Javadoc
Declaration
Console
Failure Trace

Finished after 0.15 seconds

Runs: 2/2   Errors: 0   Failures: 0

- calculatorTest [Runner: JUnit 5] (0.000 s)
  - testAdd() (0.000 s)
  - testSubtract() (0.000 s)

```java
package com.junitExe.JunitExercise1;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class calculatorTest {

    @Test
    void testAdd() {
        calculator calc = new calculator();
        int result = calc.add(2, 3);
        System.out.println("Addition Result: " + result); // 👈 this will print
        assertEquals(5, result);
    }

    @Test
    void testSubtract() {
        calculator calc = new calculator();
        int result = calc.subtract(7, 4);
        assertEquals(3, result); // ☑ this should pass
    }

}
```

```
<terminated> calculatorTest [JUnit] C:\Users\sanga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (27-Jun-2025, 3:31:01 pm – 3
Addition Result: 5
```

Exercise 3: Assertions in JUnit

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

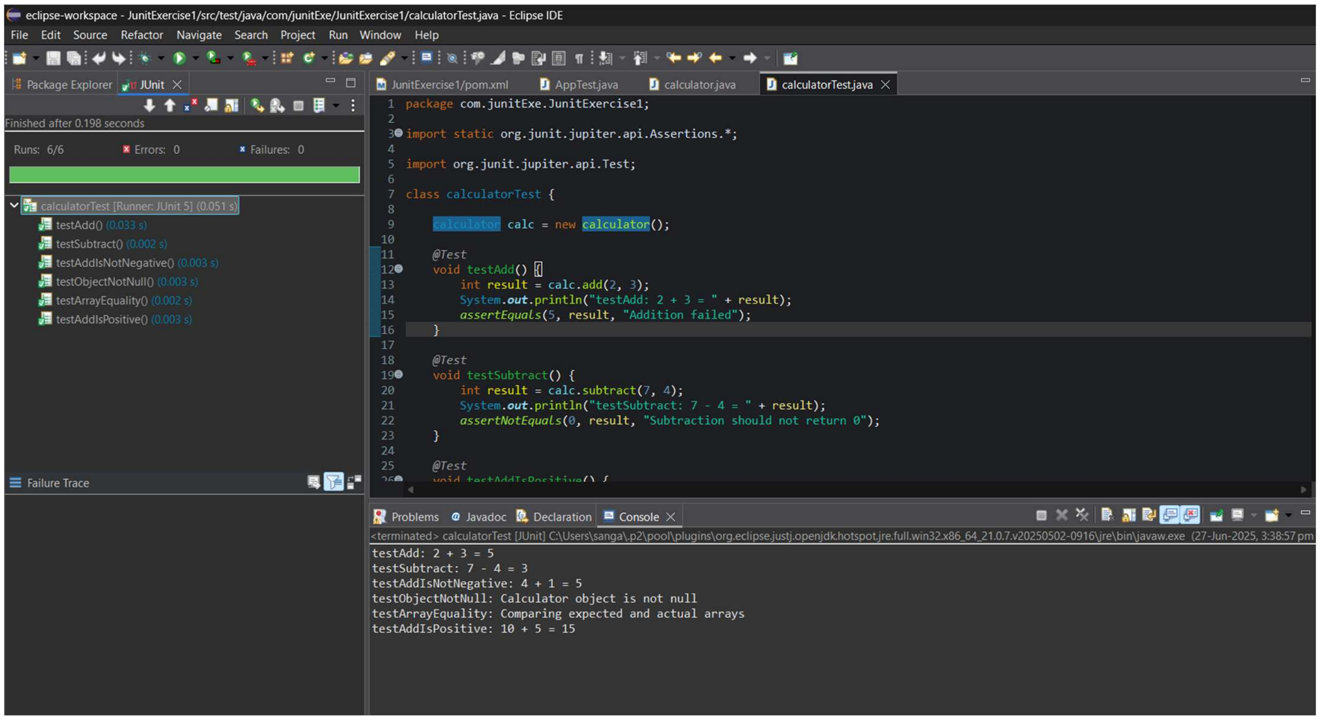
Steps:

1. Write tests using various JUnit assertions.

calculatorTest.java

```java
package com.junitExe.JunitExercise1;


import static org.junit.jupiter.api.Assertions.*;


import org.junit.jupiter.api.Test;


class calculatorTest {


        calculator calc = new calculator();


  @Test
  void testAdd() {
    int result = calc.add(2, 3);
    System.out.println("testAdd: 2 + 3 = " + result);
    assertEquals(5, result, "Addition failed");
  }


  @Test
  void testSubtract() {
    int result = calc.subtract(7, 4);
    System.out.println("testSubtract: 7 - 4 = " + result);
    assertNotEquals(0, result, "Subtraction should not return 0");
  }


  @Test
  void testAddIsPositive() {
        int result = calc.add(10, 5);
        System.out.println("testAddIsPositive: 10 + 5 = " + result);
        assertTrue(result > 0, "Result should be positive");
    }

    @Test
    void testAddIsNotNegative() {
        int result = calc.add(4, 1);
        System.out.println("testAddIsNotNegative: 4 + 1 = " + result);
        assertFalse(result < 0, "Result should not be negative");
    }

    @Test
    void testObjectNotNull() {
        System.out.println("testObjectNotNull: Calculator object is " + (calc != null ? "not null" : "null"));
        assertNotNull(calc, "Calculator object should not be null");
    }

    @Test
    void testArrayEquality() {
        int[] expected = {1, 2, 3};
        int[] actual = {1, 2, 3};
        System.out.println("testArrayEquality: Comparing expected and actual arrays");
        assertArrayEquals(expected, actual, "Arrays should be equal");
    }

}
```

Result:



eclipse-workspace - JunitExercise1/src/test/java/com/junitExe/JunitExercise1/calculatorTest.java - Eclipse IDE

Finished after 0.198 seconds

Runs: 6/6 Errors: 0 Failures: 0

```
calculatorTest [Runner: JUnit 5] (0.051 s)
    testAdd() (0.033 s)
    testSubtract() (0.002 s)
    testAddIsNotNegative() (0.003 s)
    testObjectNotNull() (0.003 s)
    testArrayEquality() (0.002 s)
    testAddIsPositive() (0.003 s)
```

```java
package com.junitExe.JunitExercise1;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class calculatorTest {

    calculator calc = new calculator();

    @Test
    void testAdd() {
        int result = calc.add(2, 3);
        System.out.println("testAdd: 2 + 3 = " + result);
        assertEquals(5, result, "Addition failed");
    }

    @Test
    void testSubtract() {
        int result = calc.subtract(7, 4);
        System.out.println("testSubtract: 7 - 4 = " + result);
        assertNotEquals(0, result, "Subtraction should not return 0");
    }

    @Test
    void testAddIsPositive() {
```

```
<terminated> calculatorTest [JUnit] C:\Users\sanga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (27-Jun-2025, 3:38:57 pm
testAdd: 2 + 3 = 5
testSubtract: 7 - 4 = 3
testAddIsNotNegative: 4 + 1 = 5
testObjectNotNull: Calculator object is not null
testArrayEquality: Comparing expected and actual arrays
testAddIsPositive: 10 + 5 = 15
```

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:

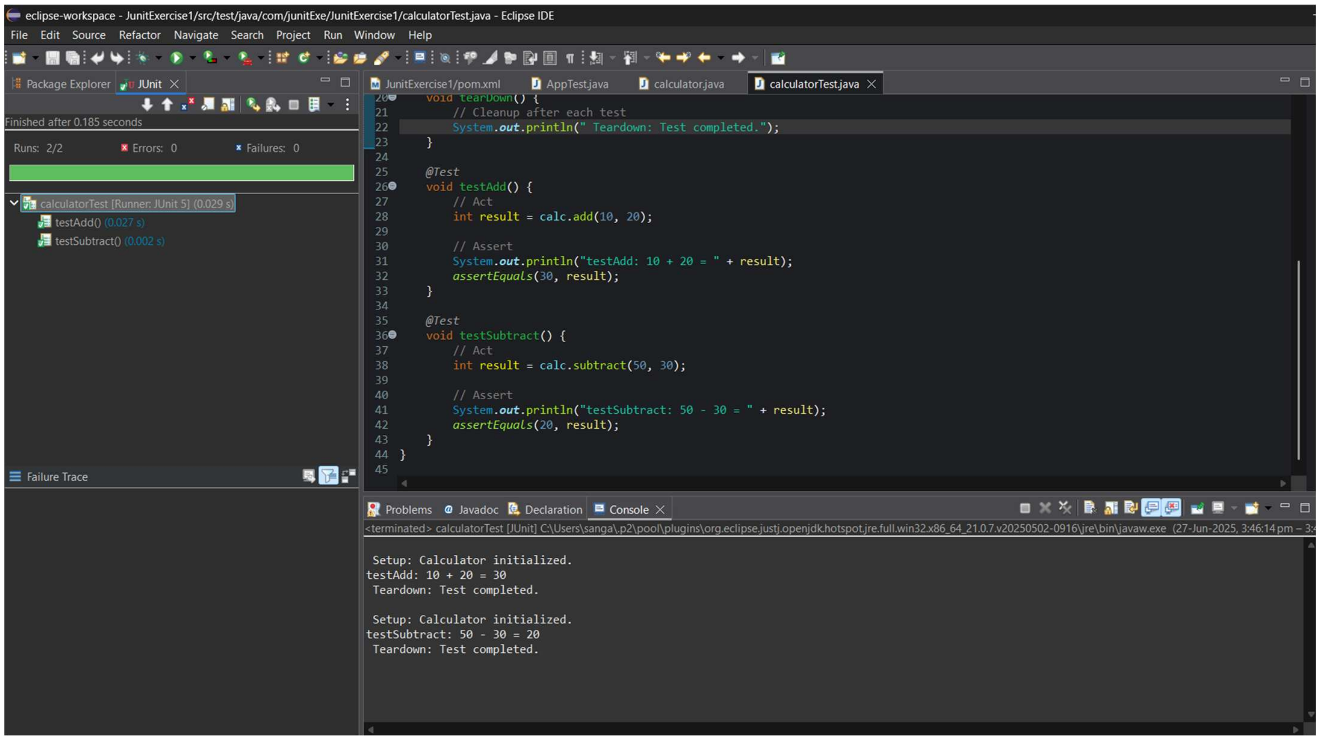
1. Write tests using the AAA pattern.

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

Solution Code:

```
package com.junitExe.JunitExercise1;


import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;


public class calculatorTest {


  calculator calc;


  @BeforeEach

  void setUp() {

    // Arrange: Initialize before each test

    calc = new calculator();

    System.out.println("\n Setup: Calculator initialized.");

  }


  @AfterEach

  void tearDown() {

    // Cleanup after each test

    System.out.println(" Teardown: Test completed.");

  }
@Test
 void testAdd() {
   // Act
   int result = calc.add(10, 20);

   // Assert
   System.out.println("testAdd: 10 + 20 = " + result);
   assertEquals(30, result);
 }

 @Test
 void testSubtract() {
   // Act
   int result = calc.subtract(50, 30);

   // Assert
   System.out.println("testSubtract: 50 - 30 = " + result);
   assertEquals(20, result);
 }
}
```

Result:


eclipse-workspace - JunitExercise1/src/test/java/com/junitExe/JunitExercise1/calculatorTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
JunitExercise1/pom.xml
AppTest.java
calculator.java
calculatorTest.java
Failure Trace
Problems
Javadoc
Declaration
Console

Finished after 0.185 seconds

Runs: 2/2 Errors: 0 Failures: 0

calculatorTest [Runner: JUnit 5] (0.029 s)
testAdd() (0.027 s)
testSubtract() (0.002 s)

```java
    void tearDown() {
        // Cleanup after each test
        System.out.println(" Teardown: Test completed.");
    }

    @Test
    void testAdd() {
        // Act
        int result = calc.add(10, 20);

        // Assert
        System.out.println("testAdd: 10 + 20 = " + result);
        assertEquals(30, result);
    }

    @Test
    void testSubtract() {
        // Act
        int result = calc.subtract(50, 30);

        // Assert
        System.out.println("testSubtract: 50 - 30 = " + result);
        assertEquals(20, result);
    }
}
```

<terminated> calculatorTest [JUnit] C:\Users\sanga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (27-Jun-2025, 3:46:14 pm – 3:

```
 Setup: Calculator initialized.
testAdd: 10 + 20 = 30
 Teardown: Test completed.

 Setup: Calculator initialized.
testSubtract: 50 - 30 = 20
 Teardown: Test completed.
```

TDD using Junit and Mockito – [Mockito Exercise]

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.

ExternalApi.java

```
package com.mockitoDemo;

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

2. Stub the methods to return predefined values.

MyService.java

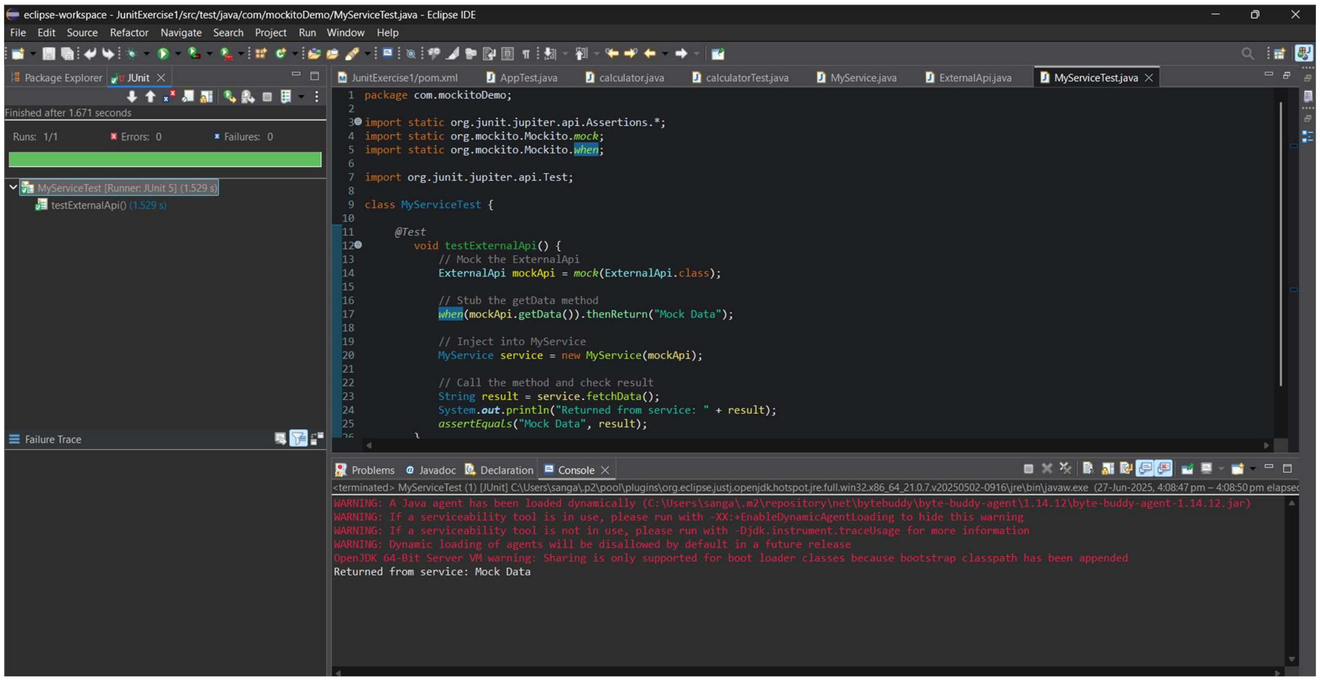
```
package com.mockitoDemo;

public class MyService {
    private ExternalApi externalApi;

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

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

3. Write a test case that uses the mock object.

MyServiceTest.java

```
package com.mockitoDemo;


import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;


import org.junit.jupiter.api.Test;


class MyServiceTest {


	@Test
	  void testExternalApi() {
	    // Mock the ExternalApi
	    ExternalApi mockApi = mock(ExternalApi.class);


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


	    // Inject into MyService
	    MyService service = new MyService(mockApi);


	    // Call the method and check result
	    String result = service.fetchData();
	    System.out.println("Returned from service: " + result);
	    assertEquals("Mock Data", result);
	  }


}
```

Result:



Finished after 1.671 seconds

Runs: 1/1 Errors: 0 Failures: 0

MyServiceTest [Runner: JUnit 5] (1.529 s)
testExternalApi() (1.529 s)

```java
package com.mockitoDemo;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import org.junit.jupiter.api.Test;

class MyServiceTest {

    @Test
        void testExternalApi() {
            // Mock the ExternalApi
            ExternalApi mockApi = mock(ExternalApi.class);

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

            // Inject into MyService
            MyService service = new MyService(mockApi);

            // Call the method and check result
            String result = service.fetchData();
            System.out.println("Returned from service: " + result);
            assertEquals("Mock Data", result);
```

```
<terminated> MyServiceTest (1) [JUnit] C:\Users\sanga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (27-Jun-2025, 4:08:47 pm – 4:08:50 pm elapsec
WARNING: A Java agent has been loaded dynamically (C:\Users\sanga\.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
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Returned from service: Mock Data
```

Explanation for Warning:

- Mockito uses **ByteBuddy** (a bytecode manipulation library).
- ByteBuddy temporarily loads a **Java agent** to allow Mockito to mock classes and interfaces.
- The JVM warns that this is **dynamic agent loading**.
- It's safe — but in future Java versions, dynamic loading **might be blocked by default** unless explicitly allowed.
- **These warnings don't affect test behaviour or correctness**

Exercise 2: Verifying Interactions

Scenario: You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.

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

2. Call the method with specific arguments.

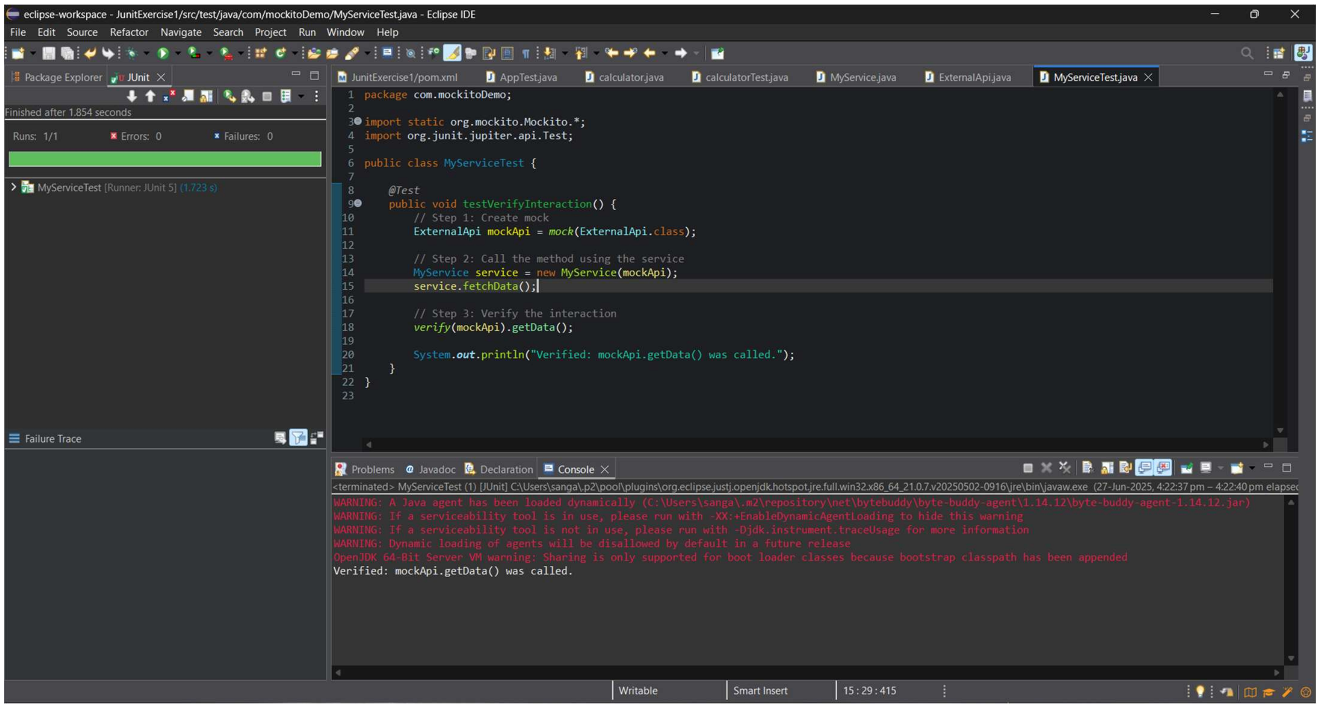
```
verify(mockApi).getData();
```

3. Verify the interaction.

MyServiceTest.java

```
package com.mockitoDemo;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

public class MyServiceTest {

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

    // Step 2: Call the method using the service
    MyService service = new MyService(mockApi);
    service.fetchData();
// Step 3: Verify the interaction
    verify(mockApi).getData();


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

Result:



Finished after 1.854 seconds

Runs: 1/1 Errors: 0 Failures: 0

MyServiceTest [Runner: JUnit 5] (1.723 s)

```java
package com.mockitoDemo;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

public class MyServiceTest {

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

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

        // Step 3: Verify the interaction
        verify(mockApi).getData();

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

```

```
<terminated> MyServiceTest (1) [JUnit] C:\Users\sanga\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe  (27-Jun-2025, 4:22:37 pm – 4:22:40 pm elapsed
WARNING: A Java agent has been loaded dynamically (C:\Users\sanga\.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
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Verified: mockApi.getData() was called.
```