**JUnit Testing**

***Exercise 1****: Setting Up JUnit in a Java Project*

**Introduction**

This document outlines the steps required to set up JUnit 4 in a Java project using Maven. JUnit is a widely-used framework for unit testing Java applications. This exercise demonstrates how to configure a Maven-based Java project, add JUnit dependencies, and write a simple test case.

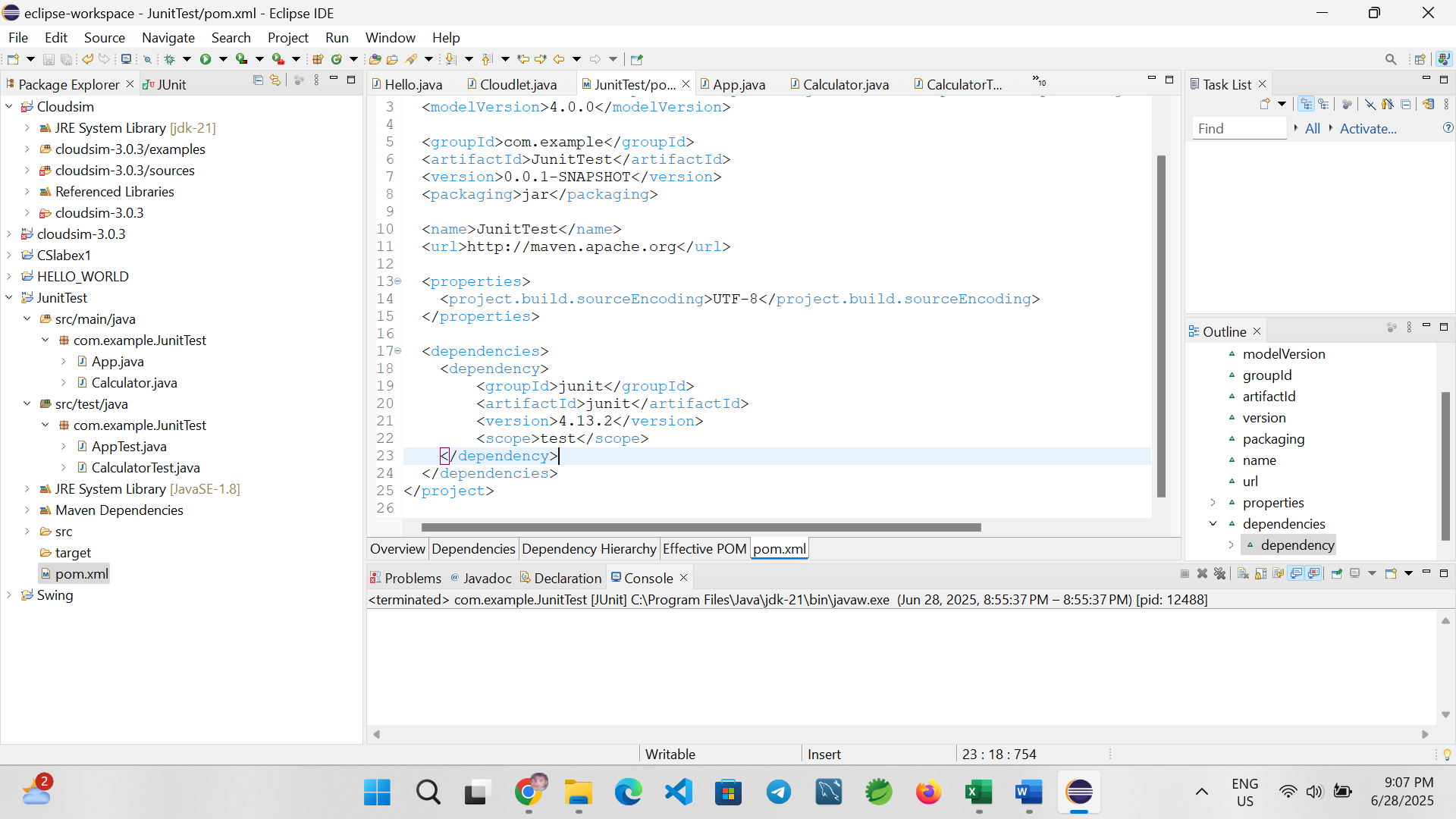
**Project Setup**

Step 1: Create a New Maven Project

* Open Eclipse → File > New > Project > Maven Project
* Use archetype: maven-archetype-quickstart
* Set Group Id: com.example
* Set Artifact Id: JUnitDemo
* Finish setup

Step 2: Add JUnit Dependency

In the pom.xml add the following dependencies



**S**tep 4: Create java class named as Calculator.java

**package** com.example.JunitTest;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

}

Step 4: Create Test class named as CalculatorTest.java

**package** com.example.JunitTest;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** CalculatorTest {

@Test

**public** **void** testAdd() {

Calculator calc = **new** Calculator();

**int** result = calc.add(2, 3);

*assertEquals*(5, result);

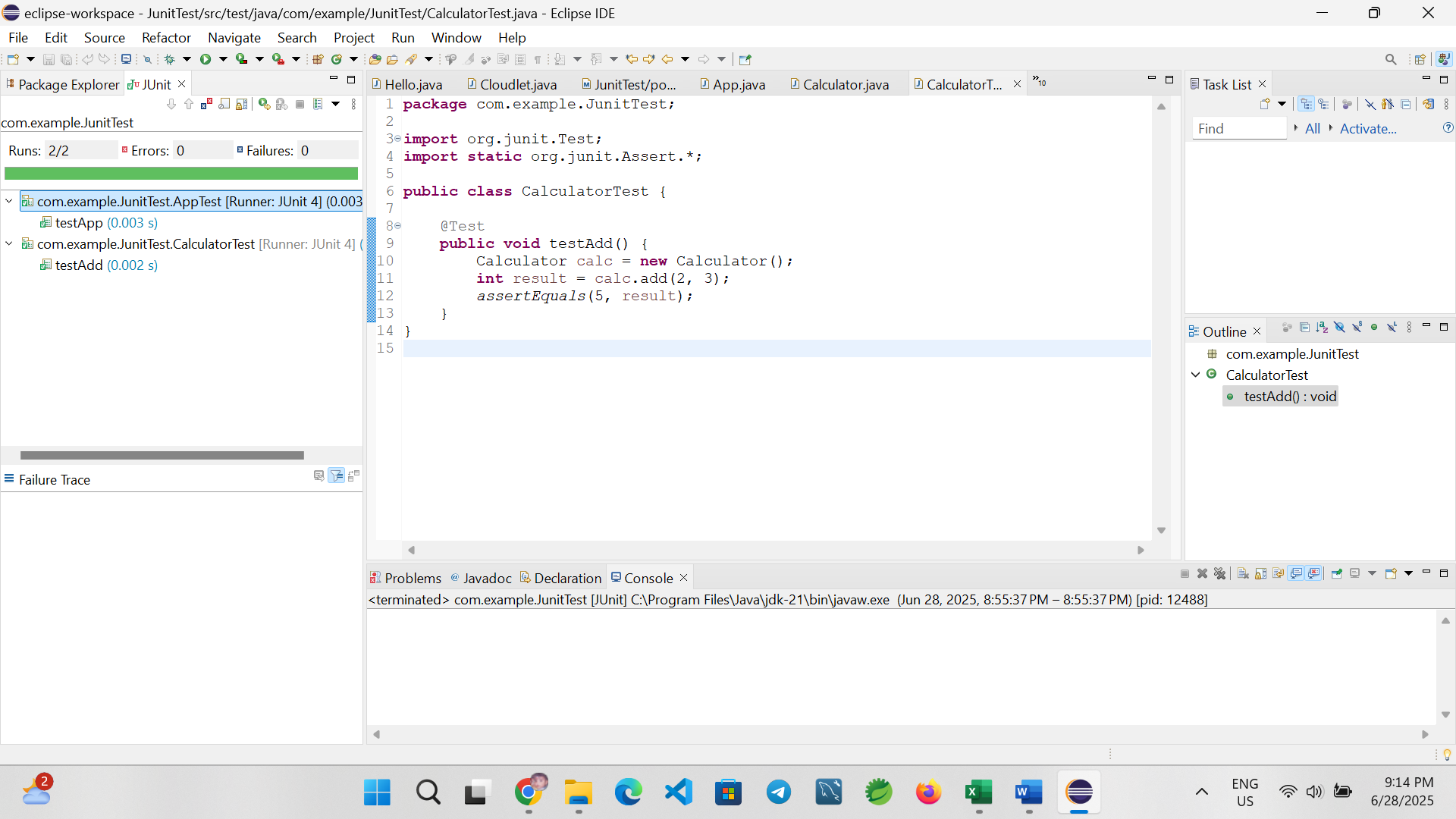
}

}

Step 5: Running the test

* Right-click on CalculatorTest.java → Run As > JUnit Test
* You should see a green bar indicating that the test passed successfully ✅

Output:



**Exercise 3: Assertions in Junit**

**Introduction**

This exercise demonstrates how to use various **JUnit assertions** to validate test results in Java. Assertions are essential for checking expected vs actual outcomes during unit testing.

**Project Setup**

Step 1: Create a New Maven Project

* Open Eclipse → File > New > Project > Maven Project
* Use archetype: maven-archetype-quickstart
* Set Group Id: com.example
* Set Artifact Id: JUnitDemo
* Finish setup

Step 2: Add dependencies in the pom.xml

Created a Maven-based Java project named AssertionsDemo and added the JUnit 4.13.2 dependency in the pom.xml.

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**C**reate a test class using Junit assertion

**package** com.example.JunitTest;

import org.junit.Test;

import static org.junit.Assert.\*;

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

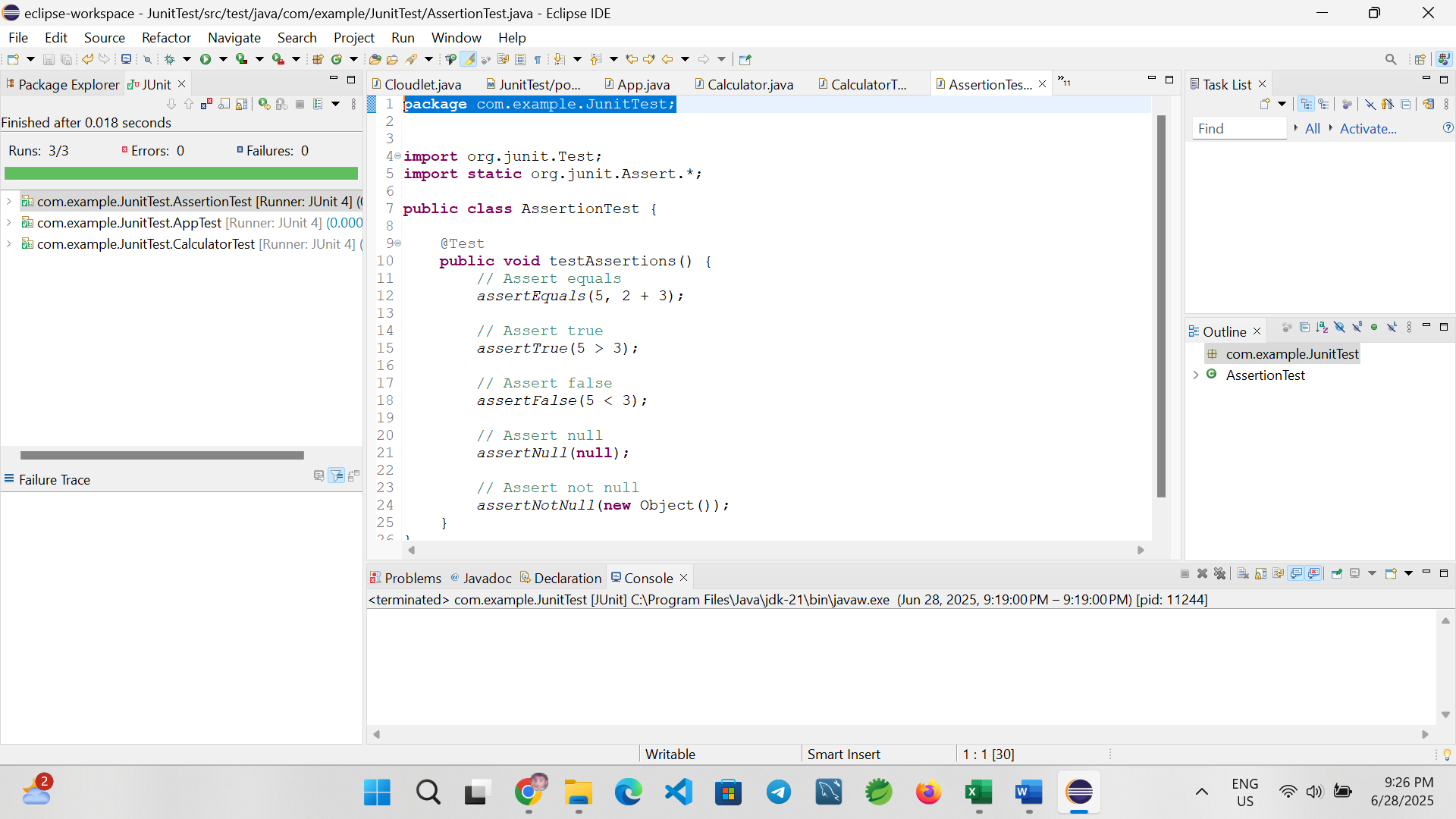
assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}



**Conclusion**

This exercise successfully demonstrates the use of various JUnit assertions including assertEquals, assertTrue, assertFalse, assertNull, and assertNotNull. These are foundational tools for writing effective unit tests.

**Exercise 4: AAA Pattern, Setup, and Teardown**

**Introduction**

This exercise demonstrates how to structure unit tests using the **Arrange-Act-Assert (AAA) pattern** and how to implement **test fixtures** in JUnit using @Before and @After annotations. These practices ensure tests are clean, reusable, and easy to maintain.

**Project Setup**

**Step 1: Create Maven Project**

* Open **Eclipse** → File > New > Maven Project
* Select archetype: maven-archetype-quickstart
* Group ID: com.example
* Artifact ID: AaaPatternDemo
* Finish

Step 2: Add JUnit Dependency in pom.xml

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

JUnit Test Class with AAA Pattern

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calc;

@Before

public void setUp() {

// Setup (runs before every test)

calc = new Calculator();

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

}

@After

public void tearDown() {

// Teardown (runs after every test)

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

}

@Test

public void testAdd() {

// Arrange

int a = 5, b = 3;

// Act

int result = calc.add(a, b);

// Assert

assertEquals(8, result);

}

@Test

public void testSubtract() {

// Arrange

int a = 10, b = 4;

// Act

int result = calc.subtract(a, b);

// Assert

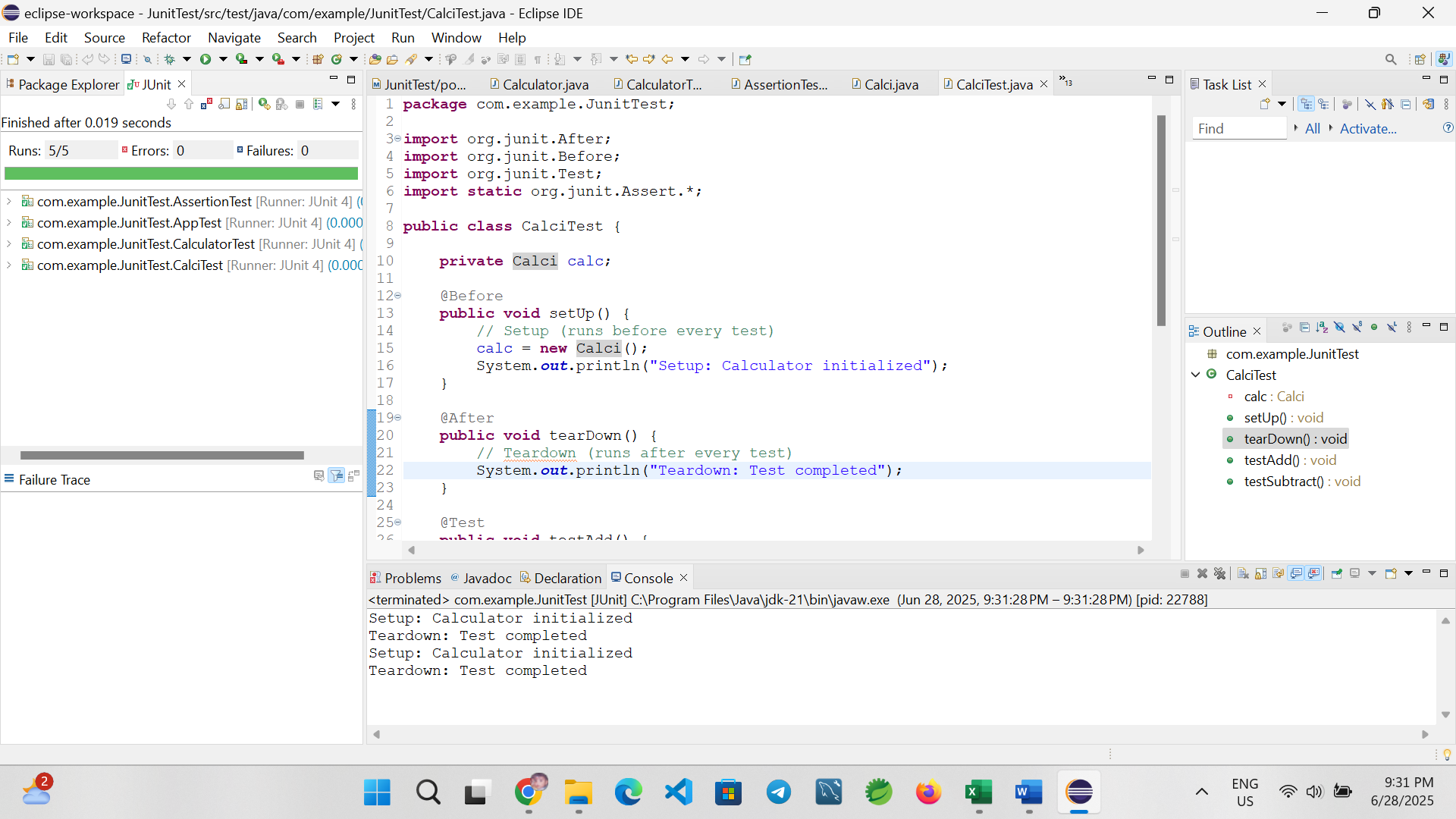
assertEquals(6, result);

}

}

**Running the Test**

* Right-click CalculatorTest.java → Run As → JUnit Test
* You should see:
  + Console output showing setup and teardown logs
  + Green bar = test passed



**Conclusion**

This exercise successfully demonstrates writing organized test cases using the **Arrange-Act-Assert pattern**, and managing test fixtures with @Before and @After in JUnit. This helps isolate test logic and ensures each test starts with a clean setup.