**Exercise 1: Setting Up JUnit**

**Scenario:**  
To begin writing unit tests, I needed to set up JUnit in my Java project using Maven in Eclipse.

**Solution:**

**Tools Used:**

* Eclipse IDE
* Apache Maven
* JUnit 5

**Steps Performed:**

1. Created a Maven Project in Eclipse
2. Added JUnit Dependency

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

3 .Created a Sample JUnit Test Case

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

package com.example;

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

import org.junit.jupiter.api.Test;

class CalculatorTest {

@Test

public void testAdd() {

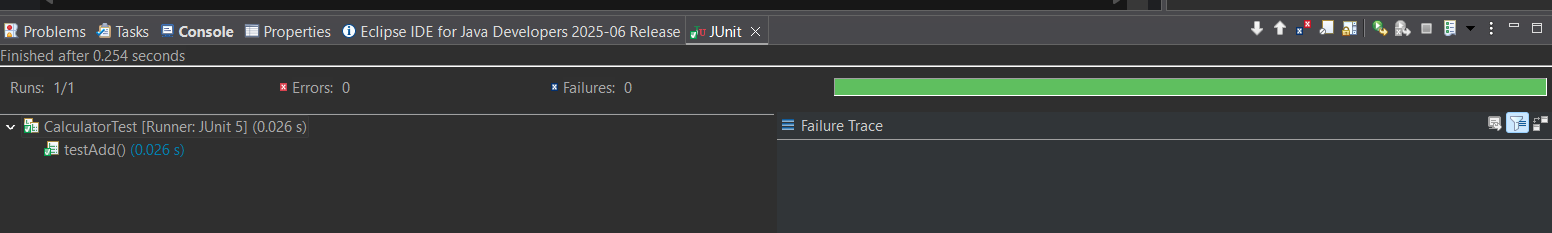
Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}

**Output:**

****

**Exercise 3: Assertions in Junit**

**Scenario:**

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

**Solution:**

**Tools Used:** Eclipse IDE, JUnit 5, Java, Maven

**Steps Performed:**

1. Created a new test class named AssertionsTest inside the src/test/java folder.
2. Wrote unit tests using different assertion methods provided by JUnit.
3. Used assertions

package com.test;

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

import org.junit.jupiter.api.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

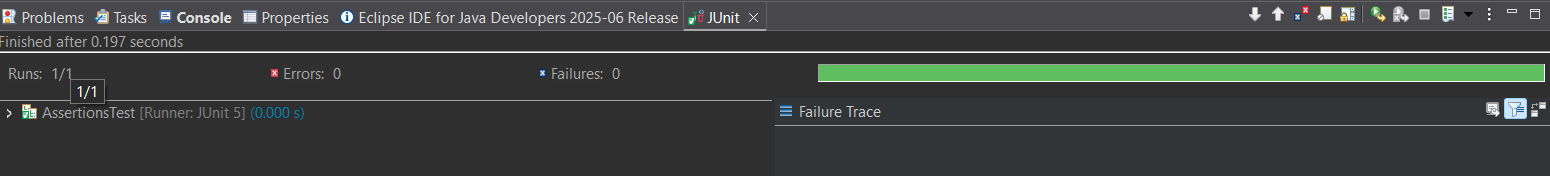
assertFalse(5 < 3);

assertNull(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.

**Solution:**

**Tools Used:** Eclipse IDE, JUnit 5, Java, Maven

**Steps Performed:**

1. Created a sample class Calculator with an add() method (in src/main/java).
2. Created a JUnit test class CalculatorTest (in src/test/java).
3. Used the AAA pattern in each test method.
4. Used @BeforeEach to initialize the Calculator object.

package com;

public class CalculatorTt {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Division by zero");

}

return a / b;

}

}

package com;

import org.junit.jupiter.api.\*;

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

public class CalculatorTtTest {

private CalculatorTt calculator;

*@BeforeEach*

void setUp() {

calculator = new CalculatorTt();

}

*@AfterEach*

void tearDown() {

calculator = null;

}

*@Test*

void testAddition() {

int result = calculator.add(2, 3);

*assertEquals*(5, result, "2 + 3 should equal 5");

}

*@Test*

void testSubtraction() {

int result = calculator.subtract(10, 4);

*assertEquals*(6, result, "10 - 4 should equal 6");

}

*@Test*

void testMultiplication() {

int result = calculator.multiply(3, 5);

*assertEquals*(15, result, "3 \* 5 should equal 15");

}

*@Test*

void testDivision() {

int result = calculator.divide(10, 2);

*assertEquals*(5, result, "10 / 2 should equal 5");

}

*@Test*

void testDivisionByZero() {

Exception exception = *assertThrows*(IllegalArgumentException.class, () -> {

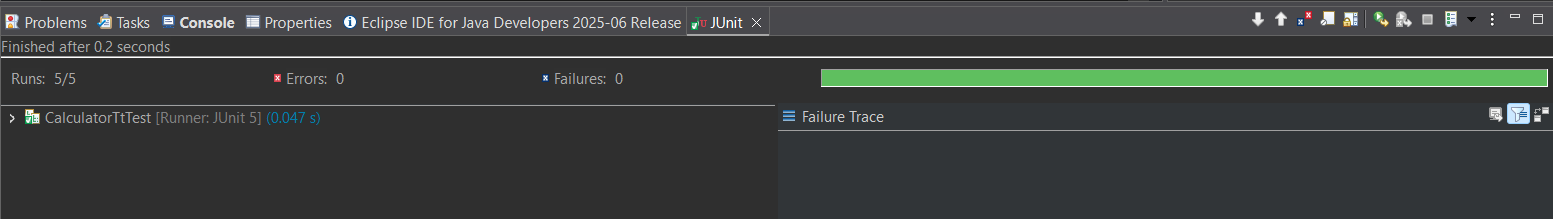
calculator.divide(10, 0);

});

*assertEquals*("Division by zero", exception.getMessage()); }

}

**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.

**Solution:**

**Tools Used:** Java 8, JUnit 5, Mockito 3.12.4, Eclipse IDE, Maven for dependency management

**Steps Performed:**

1. **Create a Maven project** in Eclipse named mockito-demo
2. **Add dependencies** in pom.xml
3. Create an external API interface

package com.example;

public interface WeatherApi {

String getWeather(String city);

}

package com.example;

public class WeatherService {

private WeatherApi weatherApi;

public WeatherService(WeatherApi weatherApi) {

this.weatherApi = weatherApi;

}

public String getWeatherMessage(String city) {

String weather = weatherApi.getWeather(city);

return "The weather in " + city + " is " + weather;

}

}

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class WeatherServiceTest {

*@Test*

public void testGetWeatherMessage() {

WeatherApi mockApi = *mock*(WeatherApi.class);

*when*(mockApi.getWeather("Chennai")).thenReturn("Sunny");

WeatherService service = new WeatherService(mockApi);

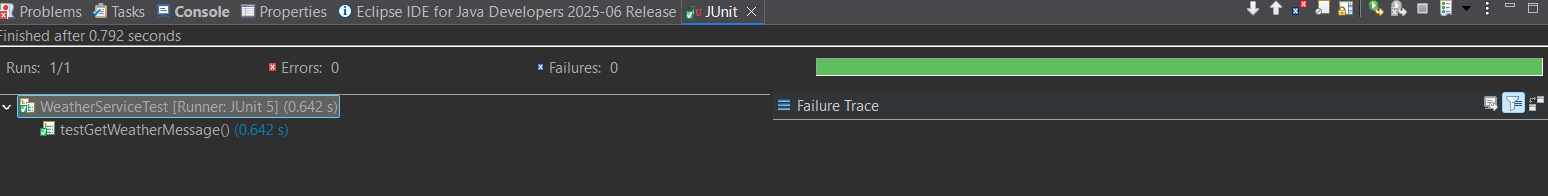
String result = service.getWeatherMessage("Chennai");

*assertEquals*("The weather in Chennai is Sunny", result);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

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

**Solution:**

**Tools Used:** Java 8, JUnit 5, Mockito 3.12.4, Eclipse IDE, Maven for dependency management

**Steps Performed:**

1. Created a Maven project and added the required dependencies (junit-jupiter and mockito-core).
2. Defined an external API interface named ExternalApi.
3. Created a service class named MyService that depends on ExternalApi.
4. Wrote a JUnit test to verify the interaction between the service and the API mock.

package New;

public interface ExternalApi {

void getData();

}

package New;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void fetchData() {

externalApi.getData();

}

}

package New;

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);

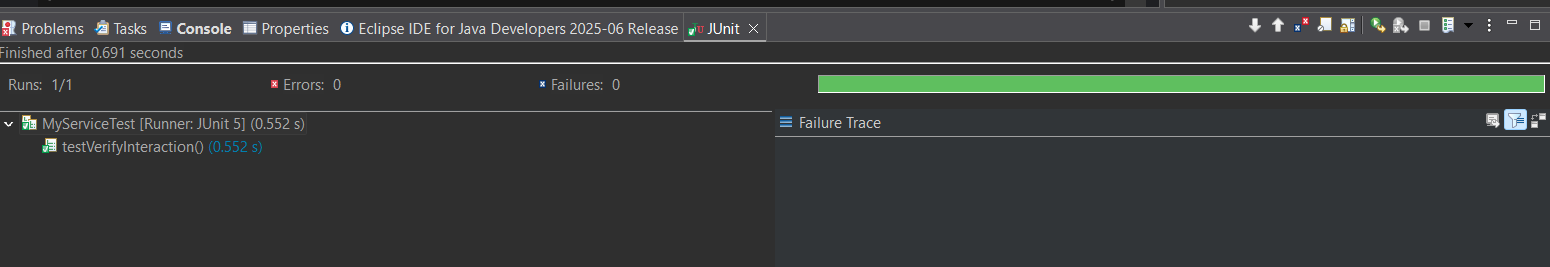
service.fetchData();

*verify*(mockApi).getData();

}

}

**Output:**

****