**WEEK – 2 HANDS ON**

**SKILL – TDD using JUnit5 and Mockito**

**JUnit – Basic Testing Exercise:**

**Exercise 1: Setting Up JUnit**

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

**CODE:**

**//Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>

    <artifactId>JUnitTesting</artifactId>

    <version>1.0-SNAPSHOT</version>

    <properties>

        <maven.compiler.source>1.8</maven.compiler.source> <!-- or use 17/23 if supported -->

        <maven.compiler.target>1.8</maven.compiler.target>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    </properties>

    <dependencies>

        <!-- JUnit dependency -->

        <dependency>

            <groupId>junit</groupId>

            <artifactId>junit</artifactId>

            <version>4.13.2</version>

            <scope>test</scope>

        </dependency>

        <!-- Hamcrest for better assertions (optional) -->

        <dependency>

            <groupId>org.hamcrest</groupId>

            <artifactId>hamcrest</artifactId>

            <version>2.2</version>

            <scope>test</scope>

        </dependency>

    </dependencies>

</project>

**//MyClass**

package org.example;

public class MyClass {

    public int add(int a, int b) {

        return a + b;

    }

}

**//MyClassTest**

package org.example;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class MyClassTest {

    @Test

    public void testAdd() {

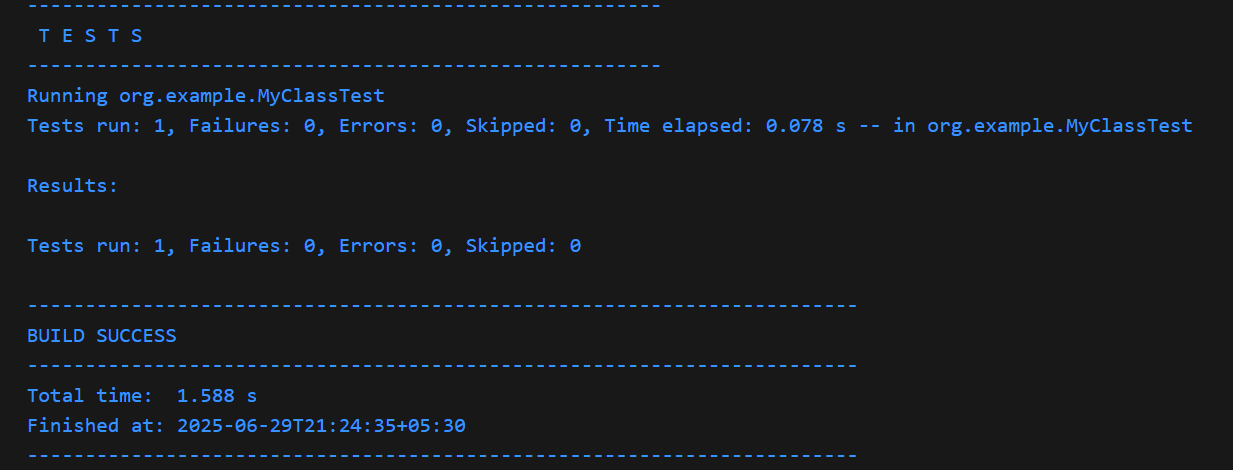
        MyClass obj = new MyClass();

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

    }

}

**OUTPUT:**



**Exercise 3: Assertions in JUnit**

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

**CODE:**

**//AssertionTest**

package org.example;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertNull;

import static org.junit.Assert.assertTrue;

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

        Object obj = null;

        assertNull(obj);

        // Assert not null

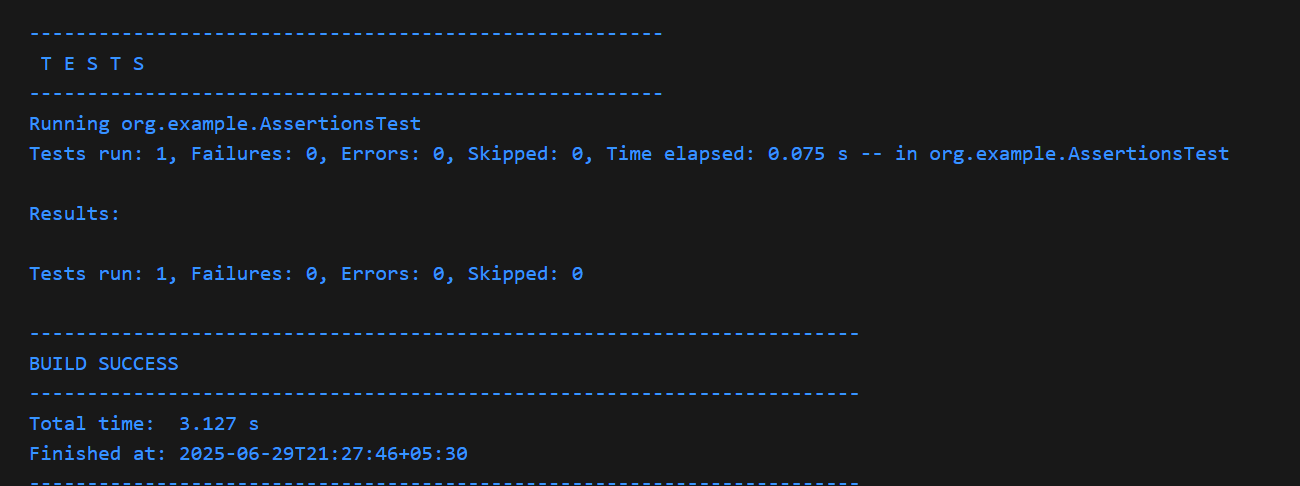
Object anotherObj = new Object();

        assertNotNull(anotherObj);

    }

}

**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.

**CODE:**

**//Calculator**

package org.example;

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int subtract(int x, int y) {

        return x - y;

    }

}

**//CalculatorTest**

package org.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

    private Calculator calculator;

    @Before

    public void setUp() {

        calculator = new Calculator();

        System.out.println("Before each test: Calculator created");

    }

    @After

    public void tearDown() {

        calculator = null;

        System.out.println("After each test: Calculator destroyed");

    }

    @Test

    public void testAdd() {

        int result = calculator.add(10, 5);

        assertEquals(15, result);

    }

    @Test

    public void testSubtract() {

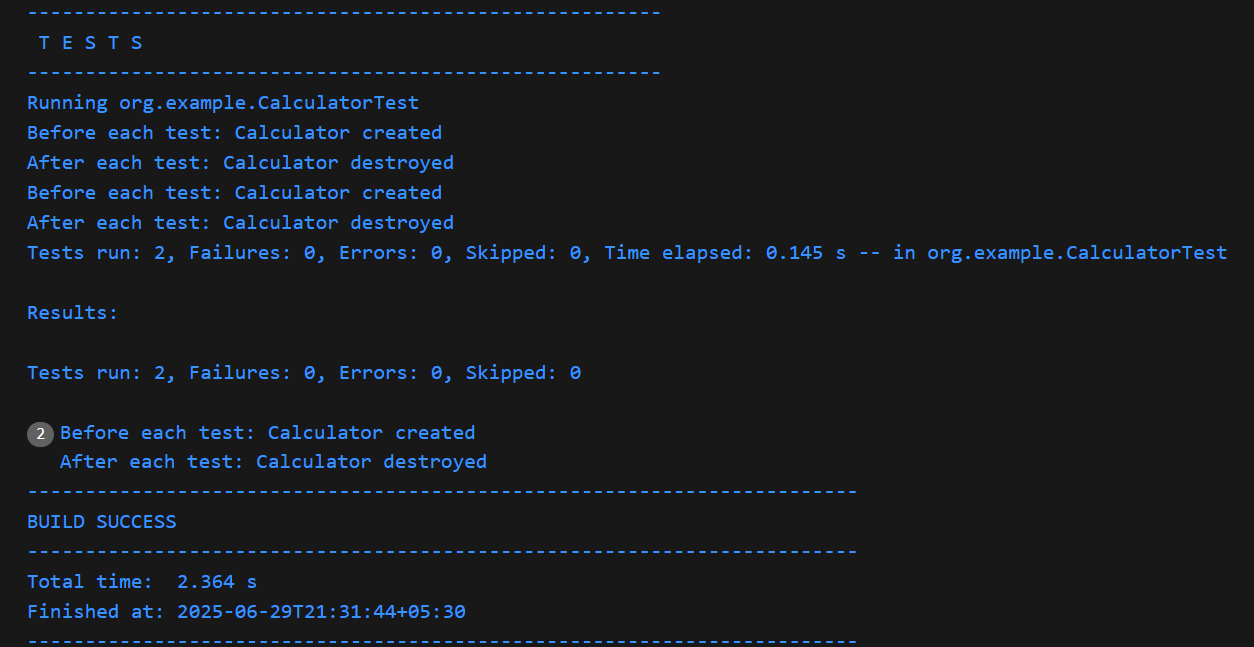
        int result = calculator.subtract(10, 3);

        assertEquals(7, result);

    }

}

**OUTPUT:**



**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.

**CODE:**

**//ExternalApi**

package org.example;

public interface ExternalApi {

    String getData();

}

**//MyService**

package org.example;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

**//MyServiceTest**

package org.example;

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() {

        ExternalApi mockApi = mock(ExternalApi.class);

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

        MyService service = new MyService(mockApi);

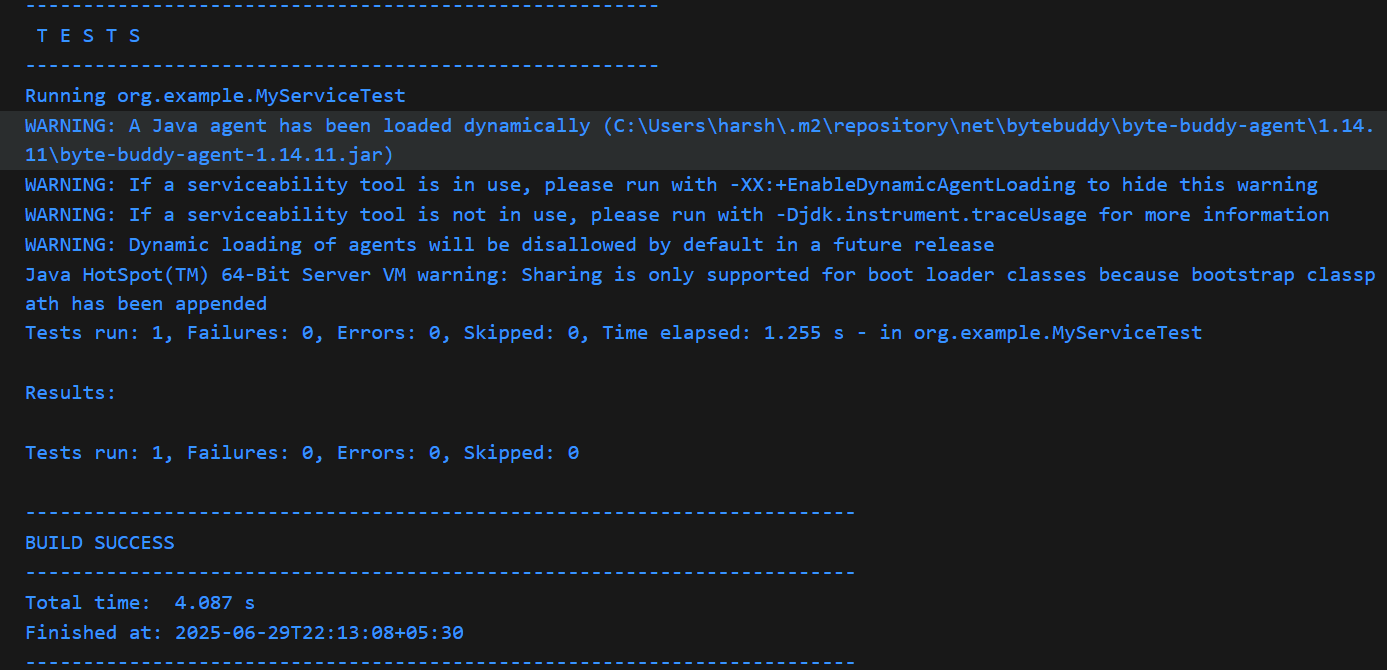
        String result = service.fetchData();

        assertEquals("Mock Data", result);

    }

}

**OUTPUT:**

****

**Exercise 2: Verifying Interactions**

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

**CODE:**

**//ExternalApi**

package org.example;

public interface ExternalApi {

    String getData();

}

**//MyService**

package org.example;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

**//MyServiceTest**

package org.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class MyServiceTest {

    @Test

    public void testVerifyInteraction() {

        ExternalApi mockApi = mock(ExternalApi.class);

        MyService service = new MyService(mockApi);

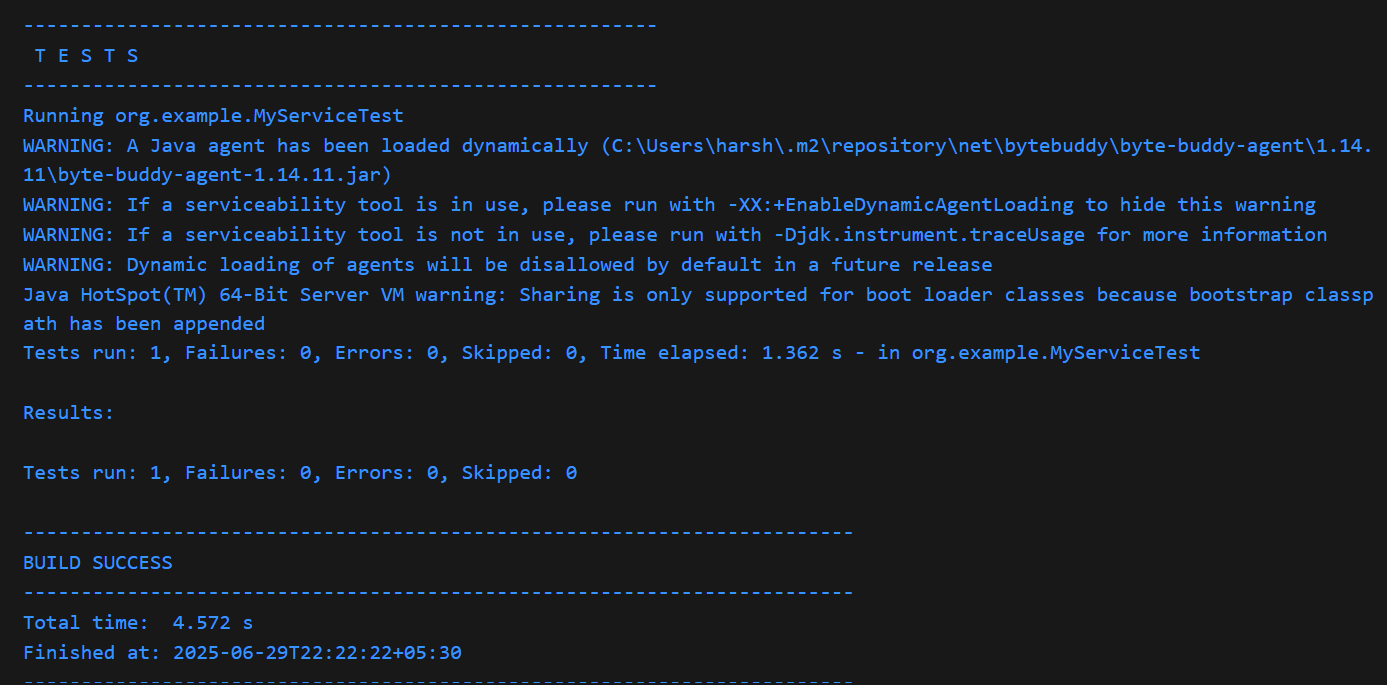
        service.fetchData();

        verify(mockApi).getData();

    }

}

**OUTPUT:**

****