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>

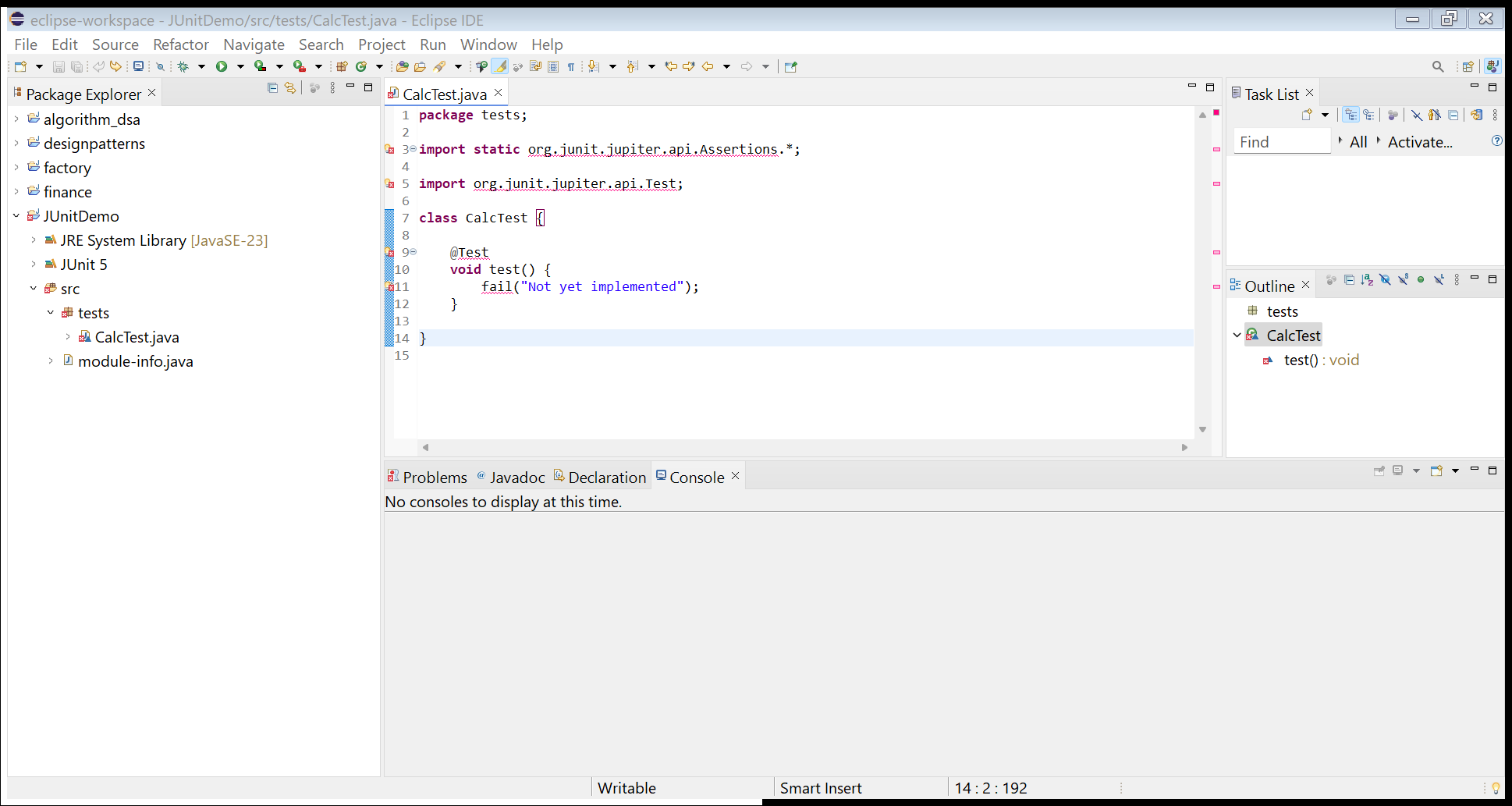
<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

Answer:



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());

}

}

Answer:

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals("Addition result should be 5", 5, 2 + 3);

assertTrue("5 should be greater than 3", 5 > 3);

assertFalse("5 should not be less than 3", 5 < 3);

assertNull("The value should be null", null);

assertNotNull("The value should not be null", new Object());

}

@Test

public void testArrayAssertions() {

int[] expectedArray = {1, 2, 3};

int[] actualArray = {1, 2, 3};

assertArrayEquals("Arrays should be equal", expectedArray, actualArray);

}

@Test

public void testSameAndNotSame() {

String str = "JUnit";

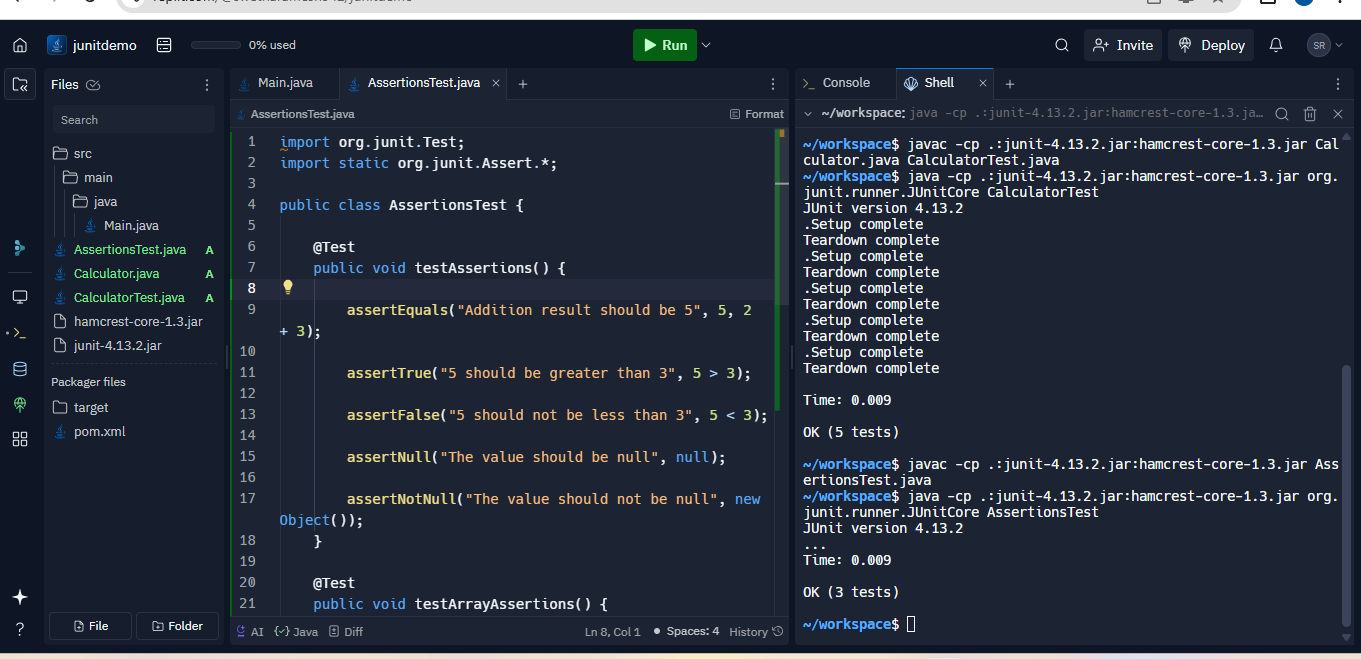
assertSame("Both references should point to the same object", str, str);

assertNotSame("References should not point to the same object", new String("JUnit"), new String("JUnit"));

}

}

Output Screenshot:



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.

Answer:

public class Calculator {

public int add(int a, int b) {

return a + b;

}

pulic 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) throws IllegalArgumentException {

if (b == 0) {

throw new IllegalArgumentException("Cannot divide by zero");

}

return a / b;

}

}

import org.junit.Before;

import org.junit.After;

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("Setup complete");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown complete");

}

@Test

public void testAdd() {

int a = 10;

int b = 5;

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

assertEquals(15, result);

}

@Test

public void testSubtract() {

int a = 10;

int b = 5;

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

assertEquals(5, result);

}

@Test

public void testMultiply() {

int a = 10;

int b = 5;

int result = calculator.multiply(a, b);

assertEquals(50, result);

}

@Test

public void testDivide() {

int a = 10;

int b = 5;

int result = calculator.divide(a, b);

assertEquals(2, result);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

int a = 10;

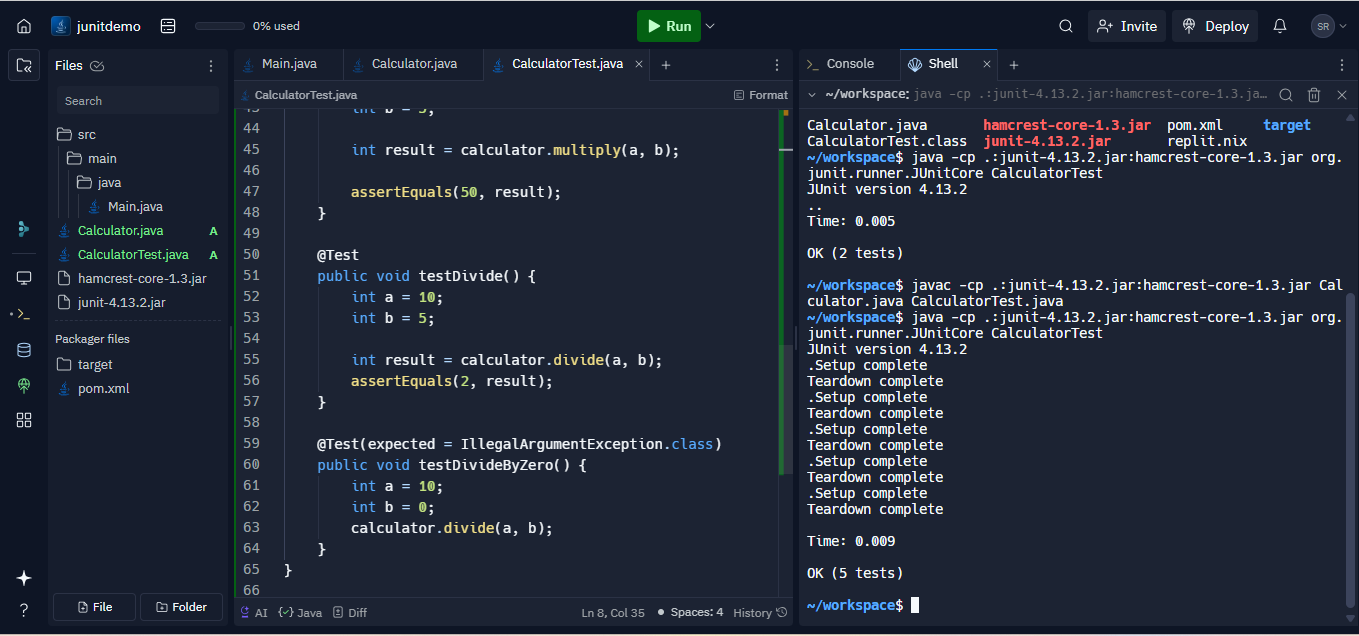
int b = 0;

calculator.divide(a, b);

}

}

Output Screenshot:



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.

Answer:

public interface ExternalApi {

String getData();

}

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testFetchData() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

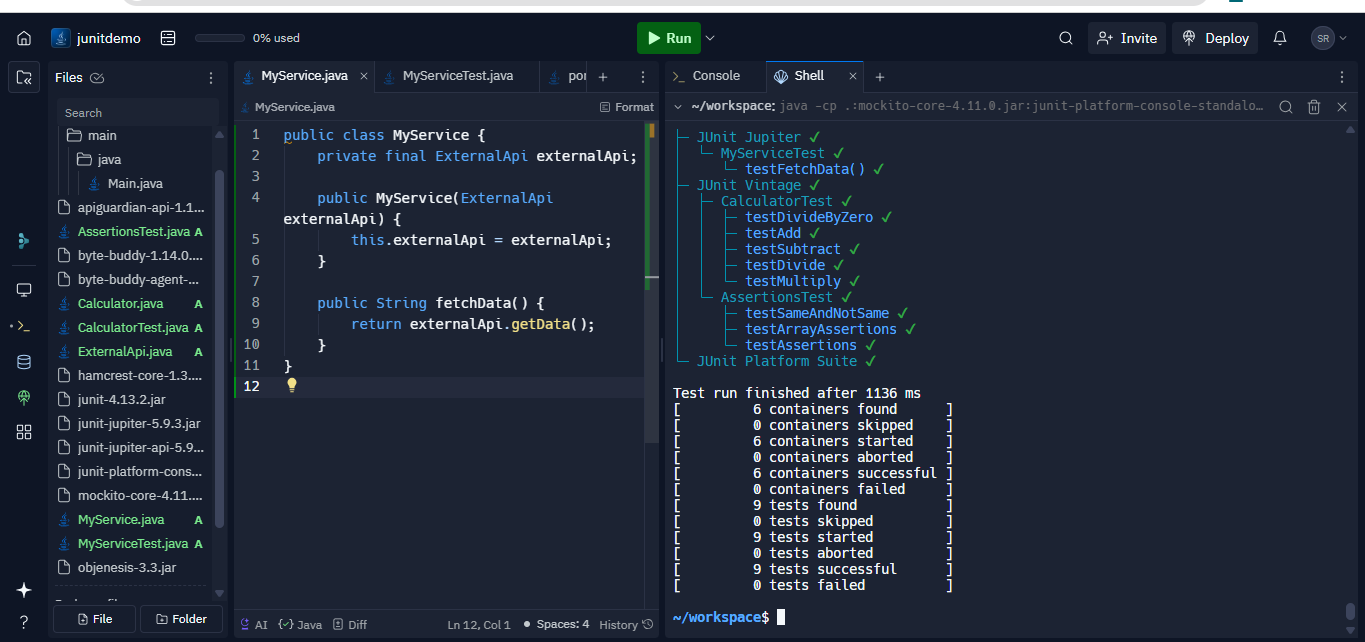
assertEquals("Mock Data", result);

verify(mockApi, times(1)).getData();

}

}

Output Screenshot:



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.

Answer:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

void getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

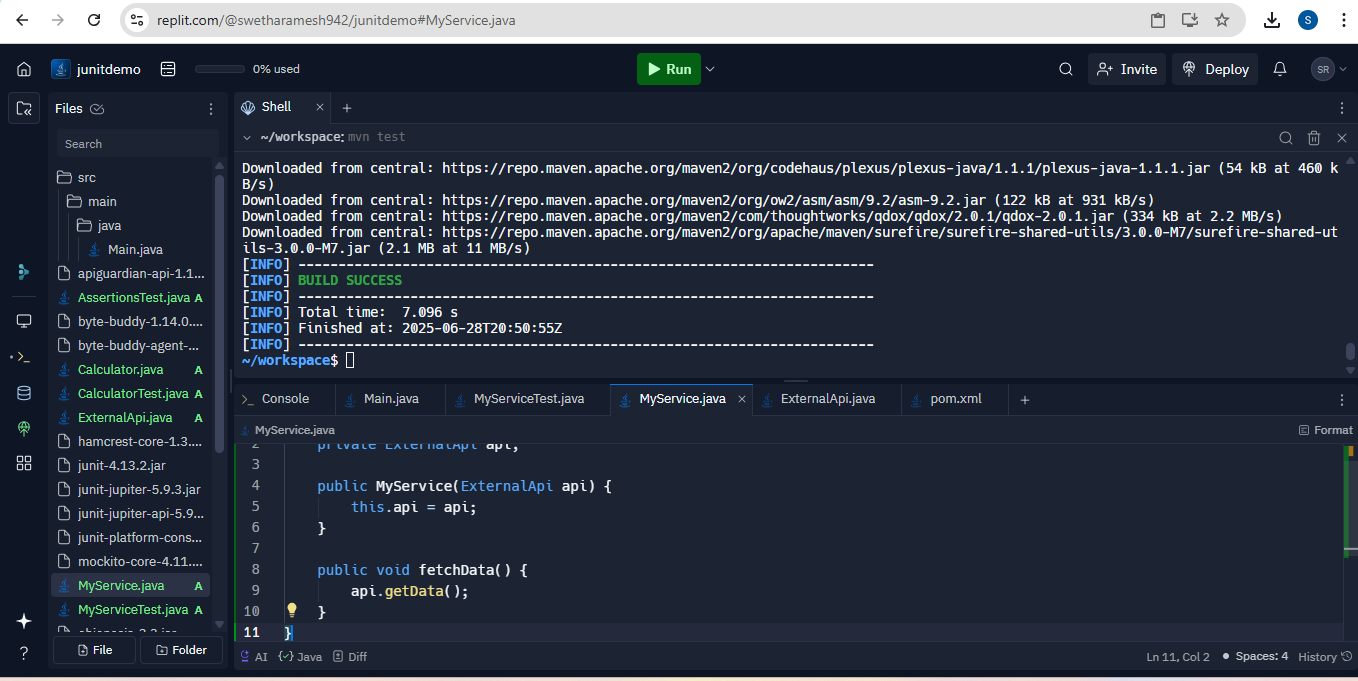
service.fetchData();

verify(mockApi).getData();

}

}

Output Screenshot:



Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels

using SLF4J.

Answer:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

logger.error("This is an error message");

logger.warn("This is a warning message");

}

}