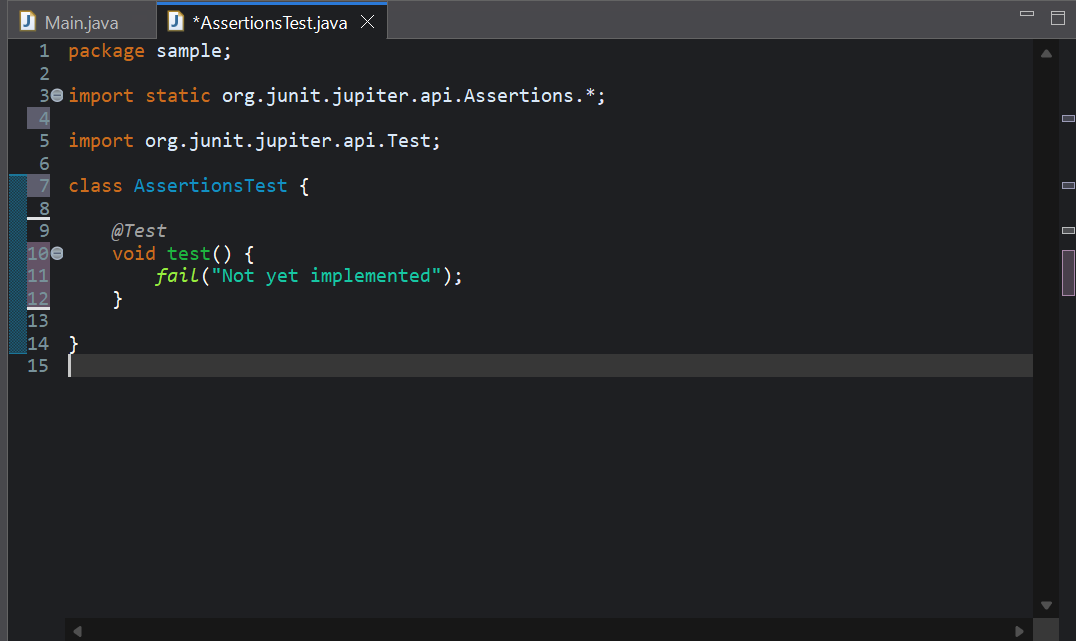
WEEK 2: TDD using JUnit5 and Mockito

**JUnit 5**

**Exercise 1: Setting Up Junit**

**Setup junit and create a new test class**

**Output:**

****

**Exercise 3: Assertions in Junit**

1. **Write tests using various JUnit assertions.**

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);

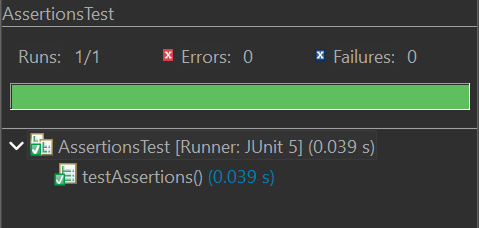
assertNull(null);

assertNotNull(new Object());

}

}

**Output:**



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Person.java**

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age < 0) {

throw new IllegalArgumentException("Age cannot be negative");

}

this.age = age;

}

}

**PersonTest.java**

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

public class PersonTest {

private Person person;

@BeforeEach

public void setUp() {

person = new Person("John", 25);

}

@AfterEach

public void tearDown() {

person = null;

}

@Test

public void testGetAge() {

// Act-Retrieve

int age = person.getAge();

// Assert

assertEquals(25, age, "The age should be 25");

}

@Test

public void testSetAge() {

// Arrange

person.setAge(30);

// Act

int age = person.getAge();

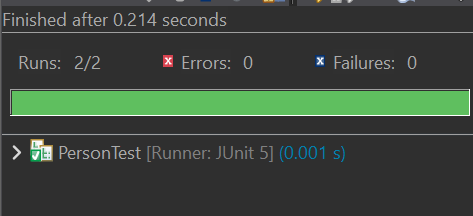
// Assert

assertEquals(30, age, "The age should be 30 after setting it to 30");

}

}

**Output:**



**Mockito**

**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

**ExternalApi.java**

package com.mypackage.mockito;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.mypackage.mockito;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**AppTest.java**

package com.mypackage.mockito;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

import org.mockito.Mockito;

public class AppTest {

@Test

public void testExternalApi() {

//mock

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

//stub

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

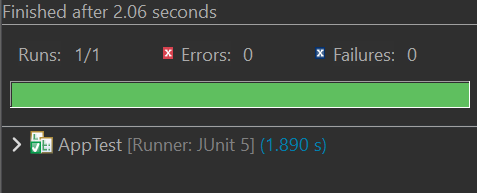
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**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.**

package com.mypackage.mockito;

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);

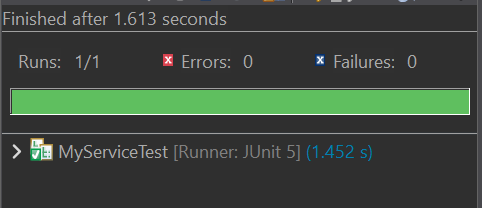
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}



**Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**Create a Java class that uses SLF4J for logging:**

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");

}

}

