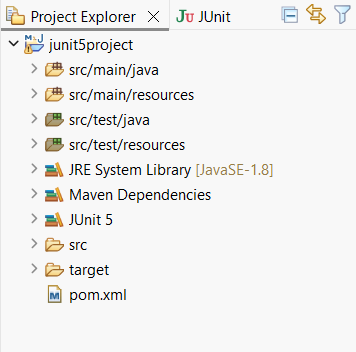
**2. JUNIT TESTING EXERCISES**

**Exercise 1: Setting Up JUnit**

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

* Create a new Maven project in your Eclipse.
* Create a simple project (skip archetype selection) -> Next.
* Group id: junit, Artifact id: junit -> Finish
* Double click pom.xml, add JUnit dependency.
* Below we can see JUnit 5 dependency.



* Create Calculator.java in src/main/java/com.example(package)

package com.example; (used in exercise 4).

public class Calculator {

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("Cannot divide by zero");

return a / b;

}

}

**Exercise 2: Writing Basic JUnit Tests**

Scenario: You need to write basic JUnit tests for a simple Java class.

* Right click on Src/main/java -> new -> Package.
* Name it as com.example.
* Right click on com.example -> New -> Class
* Name: Calculator -> finish
* Calculator.java:-

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;

}

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");

return a / b;

}

}

* Create JUnit Test Class
* Src/test/java -> new -> package name-com.example
* Right click on com.example -> test package -> new -> JUnit Test Case
* Name: CalculatorTest
* Select JUnit5 -> finish
* CalculatorTest.java

package com.example;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

void testAdd() {

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

}

@Test

void testSubtract() {

assertEquals(1, calc.subtract(3, 2));

}

@Test

void testDivide() {

assertEquals(2, calc.divide(6, 3));

}

@Test

void testDivideByZero() {

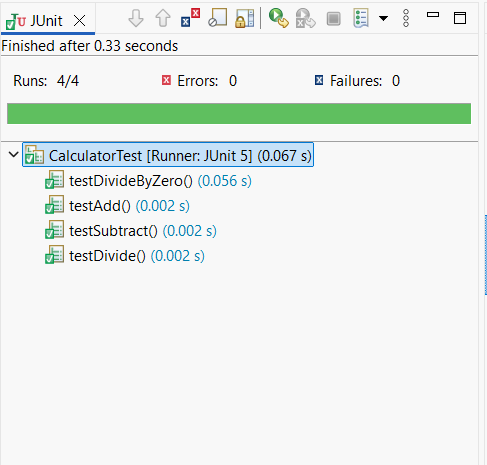
Exception e = assertThrows(IllegalArgumentException.class, () -> calc.divide(10, 0));

assertEquals("Cannot divide by zero", e.getMessage());

}

}

* CalculatorTest.java -> Run as -> JUnit Test



**Exercise 3: Assertions in JUnit**

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

* Src/test/java -> new -> package name-com.example
* Right click on com.example -> test package -> new -> JUnit Test Case
* Name: AssertionsTest
* Select JUnit5 -> finish
* AssertionsTest.java

package com.example;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(3 > 5);

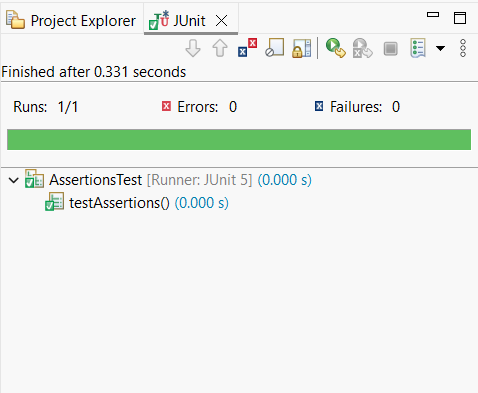
assertNull(null);

assertNotNull(new Object());

}

}

* AssertionsTest.java -> Run as -> JUnit Test

****

**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.

* Src/test/java -> new -> package name-com.example
* Right click on com.example -> test package -> new -> JUnit Test Case
* Name: CalculatorTest
* Select JUnit5 -> finish
* CalculatorTest.java

package com.example;

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

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

public class CalculatorTest {

private Calculator calc;

@BeforeEach

void setUp() {

calc = new Calculator(); // Arrange

System.out.println("Setup completed.");

}

@AfterEach

void tearDown() {

calc = null;

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

}

@Test

void testAdd() {

int result = calc.add(10, 20);

assertEquals(30, result);

}

@Test

void testSubtract() {

int result = calc.subtract(50, 20);

assertEquals(30, result);

}

@Test

void testMultiply() {

int result = calc.multiply(5, 6);

assertEquals(30, result);

}

@Test

void testDivide() {

int result = calc.divide(60, 2);

assertEquals(30, result);

}

@Test

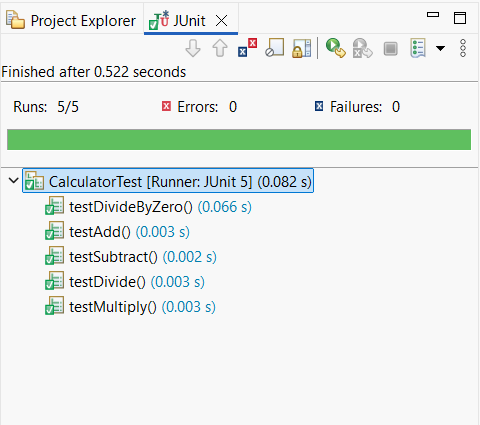
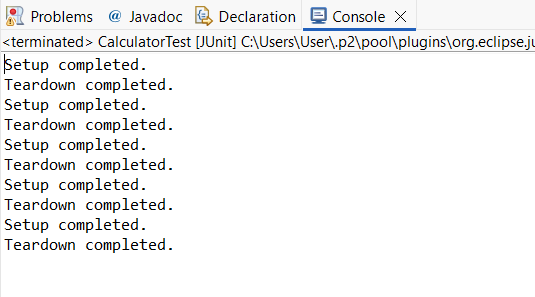
void testDivideByZero() {

Exception e = assertThrows(IllegalArgumentException.class, () -> calc.divide(5, 0));

assertEquals("Cannot divide by zero", e.getMessage());

}

}

**MOCKITO HANDS-ON EXERCISES**

**Exercise 1: Mocking and Stubbing**

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

* Eclipse -> File -> New -> Project -> Maven project -> Next.
* Click on create a simple project
* Click next.
* Group id: com.example, Artifact id: MockitoDemo -> Finish.
* Add dependency in pom.xml

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

* Src/main/java -> create package (com.example) -> interface (ExternalApi)
* ExternalApi.java

package com.example;

public interface ExternalApi {

String getData();

}

* In same package add class MyService

MyService.java

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

* Src/test/java -> package(com.example) -> class (MyServiceTest)
* MyServiceTest

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

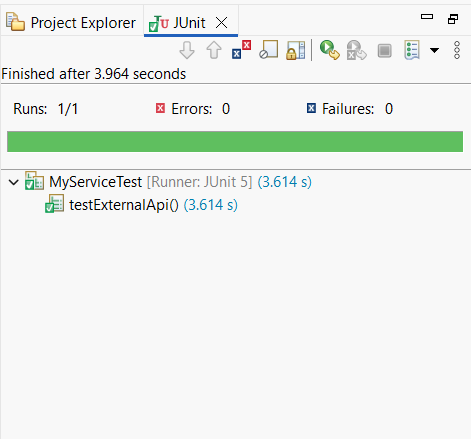
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

* Right click on MyServiceTest.java
* Click Run As -> JUnit Test



**Exercise 2: Verifying Interactions**

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

* Eclipse -> File -> New -> Project -> Maven project -> Next.
* Click on create a simple project
* Click next.
* Group id: com.example, Artifact id: MockitoDemo -> Finish.
* Add dependency in pom.xml

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

* Src/main/java -> create package (com.example) -> interface (ExternalApi)
* ExternalApi.java

package com.example;

public interface ExternalApi {

String getData();

}

* In same package add class MyService

MyService.java

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

* Src/test/java -> package(com.example) -> class (ServiceTest)
* ServiceTest.java

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class ServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

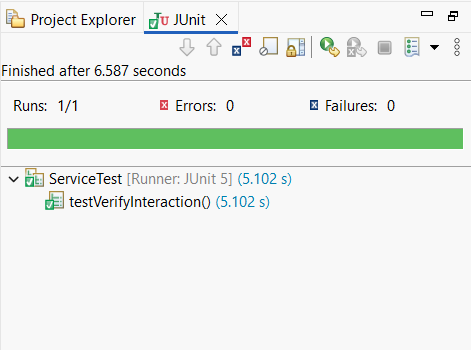
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}



**SLF4J LOGGING FRAMEWORK**

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

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

* Create Maven project in eclipse
* Add dependencies in pom.xml

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

* Src/main/java -> new -> package (com.example)
* Right clicl com.example -> new -> class (LoggingEx)
* LoggingEx.java

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingEx {

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

public static void main(String[] args) {

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

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

}

}

* Right click LoggingEx.java -> Run as -> Java Application

**Console output:**

