Week - 2

Exercise 1: 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: junit junit 4.13.2 test

3. Create a new test class in your project.

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

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

}

}

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.

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

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("Setup complete.");

}

@After

public void tearDown() {

calculator = null;

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

}

@Test

public void testAdd() {

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

assertEquals(15, result);

}

@Test

public void testMultiply() {

int result = calculator.multiply(4, 3);

assertEquals(12, result);

}

}

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.

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

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);

}

}

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

public interface ExternalApi {

String getData();

}

public class MyService {

private 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;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData(); // Verifies that getData() was called exactly once

}

}

Exercise 1: Logging Error Messages and Warning Levels Task:

Write a Java application that demonstrates logging error messages and warning levels using SLF4J. Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file

2. Create a Java class that uses SLF4J for logging: import org.slf4j.Logger

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");

logger.info("This is an informational message");

logger.debug("This is a debug message");

}

}

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>