**JUnit Testing Exercises**

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:  
<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.

Solution:

**pom.xml contents:**

xml

CopyEdit

<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>com.example</groupId>

<artifactId>JUnitDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**AppTest.java contents:**

java

CopyEdit

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class AppTest {

@Test

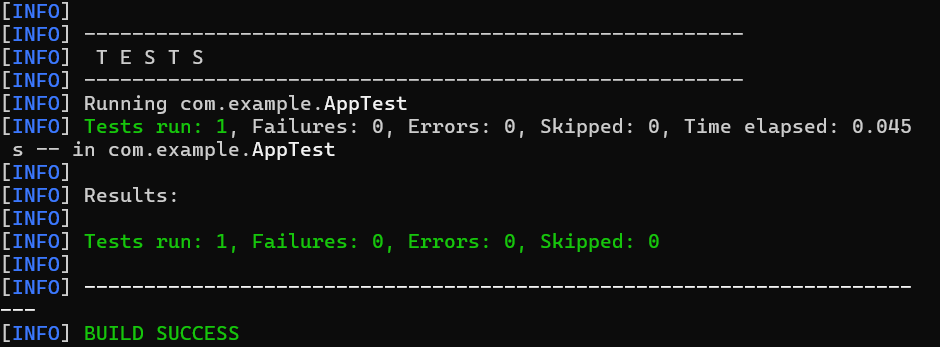
public void testAddition() {

int result = 2 + 3;

assertEquals(5, result);

}

}



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());

}

}

Solution:

**Pom.xml**

<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>com.example</groupId>

<artifactId>JUnitDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**BasicTest.java**

Save in:

D:\JUnitDemo\src\test\java\com\example\BasicTest.java

java

Copy code

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BasicTest {

@Test

public void testSum() {

int sum = 2 + 3;

assertEquals(5, sum);

}

}

**BasicTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BasicTest {

@Test

public void testSum() {

int sum = 2 + 3;

assertEquals(5, sum);

}

}

**AssertionsTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

assertNull(null);

assertNotNull(new Object());

}

}

SetupTeardo

package com.example;

import org.junit.\*;

public class SetupTeardownTest {

private int number;

@Before

public void setUp() {

number = 10;

System.out.println("Before test");

}

@Test

public void testMultiply() {

int result = number \* 2;

Assert.assertEquals(20, result);

}

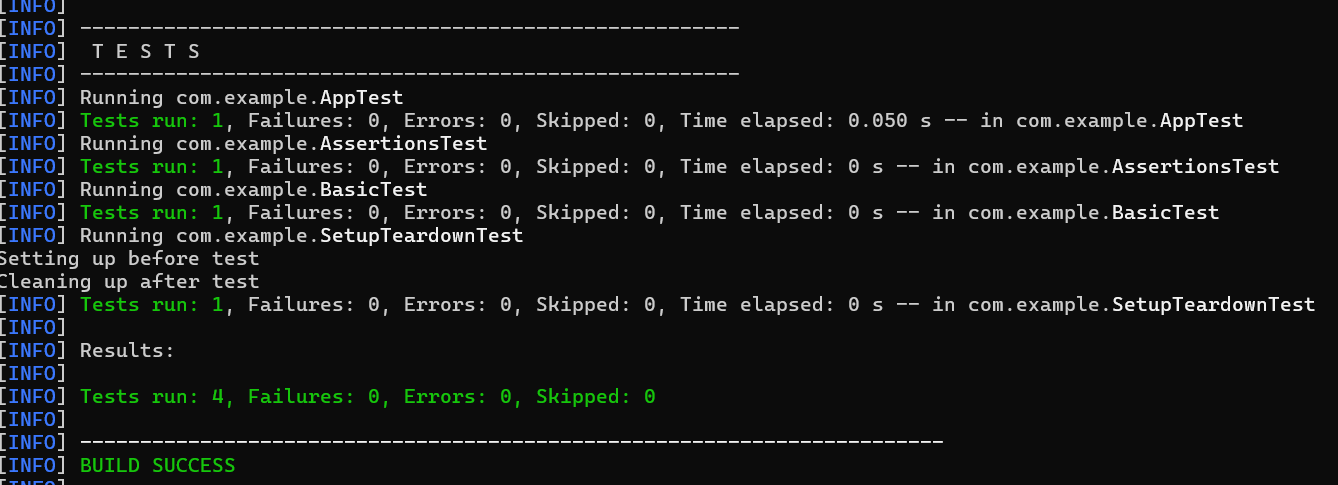
@After

public void tearDown() {

System.out.println("After test");

}

}wnTest.java



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.

Solution:

**Pom.xml**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**SetupTeardownTest.java**

package com.example;

import org.junit.\*;

public class SetupTeardownTest {

private int number;

// Arrange

@Before

public void setUp() {

number = 10;

System.out.println("Before each test – setUp");

}

// Act & Assert

@Test

public void testDoubleValue() {

int result = number \* 2;

Assert.assertEquals(20, result);

}

@Test

public void testTripleValue() {

int result = number \* 3;

Assert.assertEquals(30, result);

}

// Cleanup

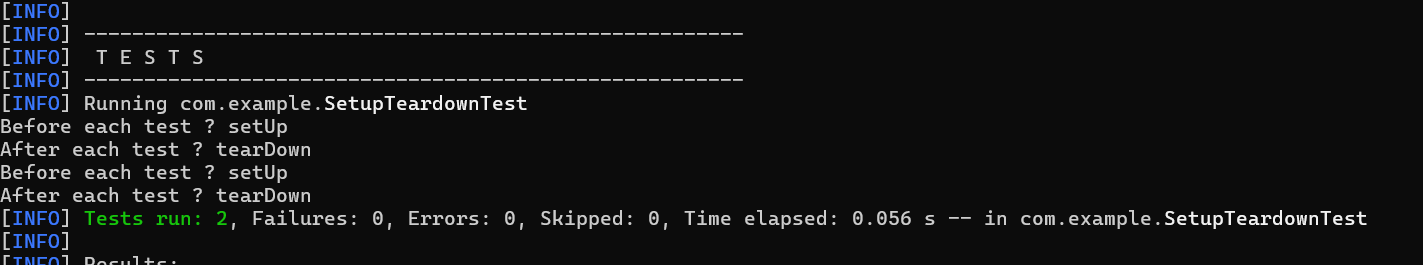
@After

public void tearDown() {

System.out.println("After each test – tearDown");

}

}



**Mockito Hands-On 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.  
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.  
Solution Code:  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
public class MyServiceTest {  
@Test  
public void testExternalApi() {  
ExternalApi mockApi = Mockito.mock(ExternalApi.class);  
when(mockApi.getData()).thenReturn("Mock Data");  
MyService service = new MyService(mockApi);  
String result = service.fetchData();  
assertEquals("Mock Data", result);  
}  
}

Solution:

ExternalApi.java

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

java

Copy code

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

java

Copy code

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

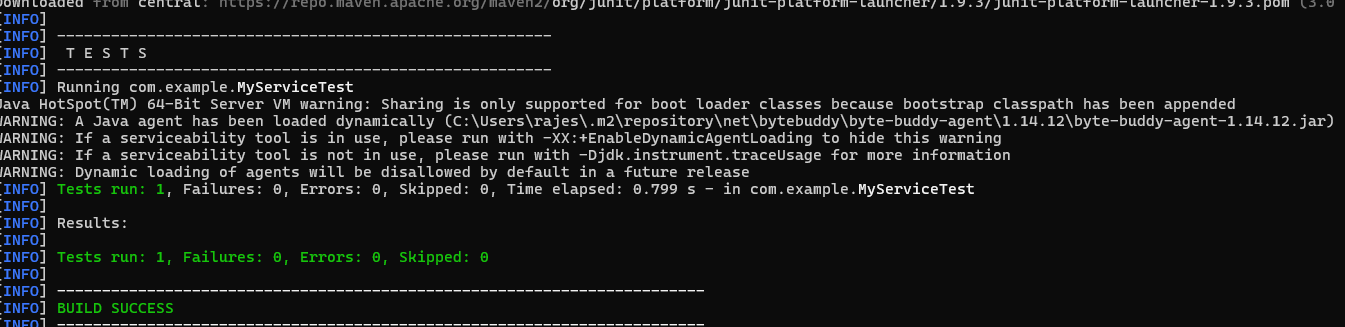
MyService service = new MyService(mockApi);

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.

Solution Code:

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);

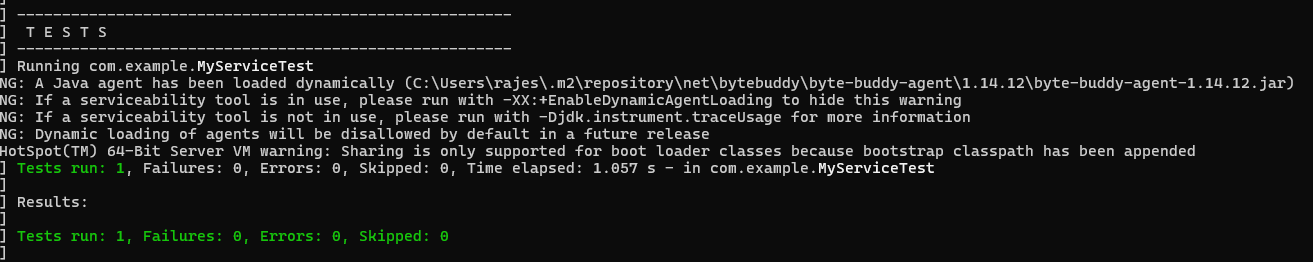
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

Output:

**Logging using SLF4J**

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: org.slf4j slf4j-api 1.7.30 ch.qos.logback logback-classic 1.2.3 2. 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"); } }

**LoggingExample.java**

package com.example;

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");

}

}

**Pom.xml**

<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>com.example</groupId>

<artifactId>LoggingDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<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>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.example.LoggingExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

logback.xml

<configuration>

<appender name="FILE" class="ch.qos.logback.core.FileAppender">

<file>logs/app.log</file>

<encoder>

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

</encoder>

</appender>

<root level="debug">

<appender-ref ref="FILE" />

</root>

</configuration>