**JUNIT BASIC TESTING**

**Exercise-1:** **Setting Up Junit**

1. Create a new Java project with name JUnitDemo in eclipse

Go to File → New → Project…

Select Maven Project → Click Next

Choose Create a simple project (skip archetype selection) → Next

Fill:

* Group Id: com.example
* Artifact Id: JUnitDemo

Click Finish

1. Adding dependency of Junit to pom.xml file to get Maven and that dependenc is

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

</dependencies>

1. Adding 2 new packages com.example, com.testing in src/main/java and src/test/java and add the following classes in them respectively

**Calci.java**

package com.example;

public class Calci {

public int add(int a, int b) {

return a + b;

}

}

**CalciTest.java**

package com.testing;

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

import org.junit.jupiter.api.Test;

class CalciTest {

@Test

void testAdd() {

Calci calc = new Calci();

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

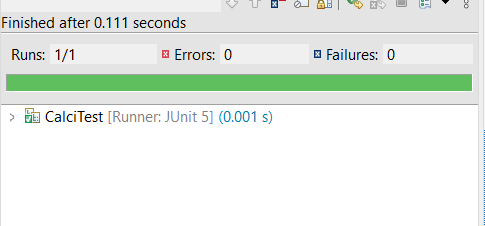
assertEquals(15, result);

}

}

1. Run the Application text class as **RunAs-> Junit Test**

Output: Junit Tab



Exercise 3: Assertions in JUnit Scenario:

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

1.In the same project and same packages add the following classes in their respective packages

MyUtils.java

package com.testing;

public class MyUtils {

public int add(int a, int b) {

return a + b;

}

public boolean isEven(int num) {

return num % 2 == 0;

}

public String getMessage() {

return "Hello";

}

public int[] getArray() {

return new int[] {1, 2, 3};

}

public void throwError() {

throw new IllegalArgumentException("Invalid input");

}

}

2.MyUtilsTest.java

package com.testing;

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

import org.junit.jupiter.api.Test;

public class MyUtilsTest {

MyUtils utils = new MyUtils();

@Test

void testAdd() {

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

}

@Test

void testIsEven() {

assertTrue(utils.isEven(4));

assertFalse(utils.isEven(5));

}

@Test

void testGetMessage() {

assertNotNull(utils.getMessage());

}

@Test

void testGetArray() {

assertArrayEquals(new int[]{1, 2, 3}, utils.getArray());

}

@Test

void testException() {

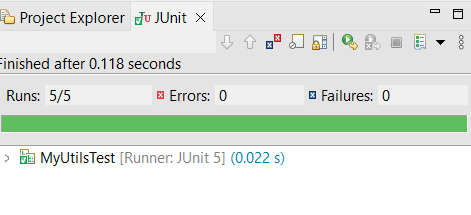
assertThrows(IllegalArgumentException.class, () -> utils.throwError());

}

}

4.Run the application test calss as Run As->Junit Test

Output: Junit Tab



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.In the same project and same packages add the following classes in their respective packages

StringUtils.java

package com.testing;

public class StringUtils {

public String toUpper(String input) {

return input.toUpperCase();

}

public boolean isEmpty(String input) {

return input == null || input.isEmpty();

}

public void clear() {

System.***out***.println("Cleared resources (if any).");

}

}

StringUtilsTest.java

package com.testing;

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

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

public class StringUtilsTest {

StringUtils utils;

@BeforeEach

void setUp() {

utils = new StringUtils();

System.out.println("Setup: StringUtils instance created");

}

@AfterEach

void tearDown() {

utils.clear()

System.out.println("Teardown: StringUtils cleared");

}

@Test

void testToUpper() {

String input = "hello";

String result = utils.toUpper(input);

assertEquals("HELLO", result);

}

@Test

void testIsEmptyWithNull() {

assertTrue(utils.isEmpty(null));

}

@Test

void testIsEmptyWithEmptyString() {

assertTrue(utils.isEmpty(""));

}

@Test

void testIsEmptyWithText() {

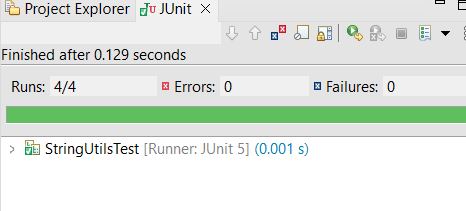
assertFalse(utils.isEmpty("abc"));

}

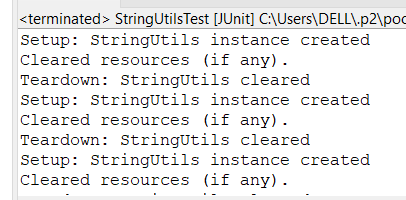
}

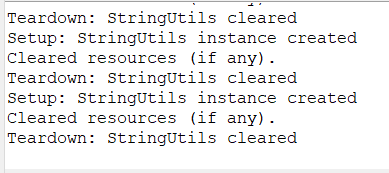
2. Run the application as Run As ->Junit Test

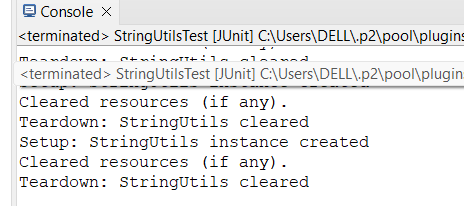
Output: Junit Tab



Console:







**Mockito Exercise**

**Exercise-1: Mocking and Stubbing**

1.Create a Maven Project with name MockitoDemo

Go to File → New → Project…

Select Maven Project → Click Next

Choose Create a simple project (skip archetype selection) → Next

Fill:

* Group Id: com.example
* Artifact Id: MockitoDemo

Click Finish

2.Add dependency for Mockito in pom.xml file as below

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>MockitoDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</project>

3. Create the package as com.example in src/main/java and add the following interface

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getInfo();

}

**MyService.java class**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getInfo();

}

}

4.Create a package com.testing in src/test/java and add following class

**MyServiceTest.java**

package com.testing;

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()

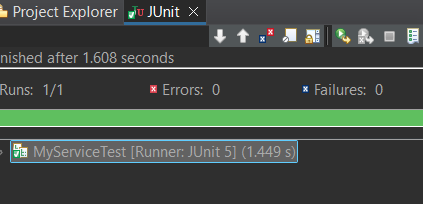
assertEquals("Mock Data", result);

}

}

4 Run the application as Run As->Junit Test

Output: Junit Tab



**Exercise 2:** Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments.

1.In the same project change the MyServiceTesting.java class

MyServiceTest.jva

package com.testing

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(); // This should internally call mockApi.getData(

verify(mockApi).getInfo();

}

}

2 Run the application as Run As->Junit Test

Output: Junit Tab

