**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.

**Code:**

package test;

import static org.junit.Assert.\*;

import org.junit.Test;

class LoanChecker {

public boolean isEligible(double salary, int creditScore) {

if (salary <= 0 || creditScore <= 0) {

throw new IllegalArgumentException("Invalid input data");

}

return (salary >= 30000) && (creditScore >= 650);

}

}

public class LoanCheckerTest {

@Test

public void testEligibleCustomer() {

LoanChecker checker = new LoanChecker();

boolean result = checker.isEligible(50000, 700);

assertTrue(result);

}

@Test

public void testIneligibleCustomer() {

LoanChecker checker = new LoanChecker();

boolean result = checker.isEligible(20000, 600);

assertFalse(result);

}

@Test(expected = IllegalArgumentException.class)

public void testInvalidInput() {

LoanChecker checker = new LoanChecker();

checker.isEligible(-1000, 500); // invalid

}

}

**Output:**

Running test.LoanCheckerTest

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESS

**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.

**Code:**

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());

}

}

**Output:**

Running AssertionsTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESS

**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.

**Code:**

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

class Calculator {

public int add(int a, int b) {

return a + b;

}

}

public class CalculatorTest {

private Calculator calc;

@Before

public void setUp() {

calc = new Calculator();

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

}

@After

public void tearDown() {

System.out.println("Test finished");

}

@Test

public void testAddition() {

int result = calc.add(2, 3);

assertEquals(5, result);

}

}

**Output:**

Running test.CalculatorTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESS

**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.

**Code:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

assertEquals("Mock Data", service.fetchData());

}

}

**Output:**

Running MyServiceTest

Tests run: 1, Failures: 0, Errors: 0

BUILD SUCCESS

**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.

**Code:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

Running MyServiceTest

Tests run: 1, Failures: 0, Errors: 0

BUILD SUCCESS