**JUNIT TESTING**

**Exercise 1:** **Setting Up JUnit**

**STEPS:** **Setting up JUnit in Eclipse.**

1. You need to set up JUnit in your Java project to start writing unit tests.
2. Open Eclipse and create a new Java project.
3. Right-click the project → Build Path → Add Libraries.
4. Select JUnit and click Next, then choose JUnit 4 and finish.
5. Create a new Java class in the src folder for your test.
6. Write test methods using @Test annotation and run with Run As → JUnit Test.

**StringUtils.java**

package testdemo;

public class StringUtils {

public static String reverse(String input) {

if (input == null) {

System.*out*.println("Input is null");

return null;

}

String reversed = new StringBuilder(input).reverse().toString();

System.*out*.println("Original: " + input + " | Reversed: " + reversed);

return reversed;

}

}

**StringUtilsTest.java**

package testdemo;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

@Test

public void testReverse\_NormalString() {

String result = StringUtils.*reverse*("hello");

System.*out*.println("Reversed 'hello': " + result);

*assertEquals*("olleh", result);

}

@Test

public void testReverse\_EmptyString() {

String result = StringUtils.*reverse*("");

System.*out*.println("Reversed empty string: '" + result + "'");

*assertEquals*("", result);

}

@Test

public void testReverse\_NullInput() {

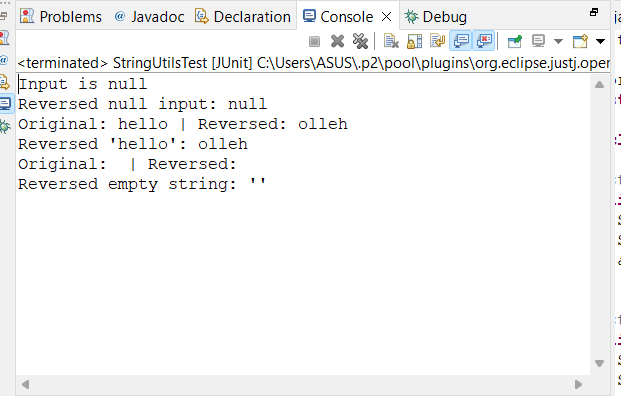
String result = StringUtils.*reverse*(null);

System.*out*.println("Reversed null input: " + result);

*assertNull*(result);

}

}

OUTPUT:  


**Exercise 3: Assertions in Junit**

**AssertionsTest.java**

package testdemo2;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

*assertNull*(null);

*assertNotNull*(new Object());

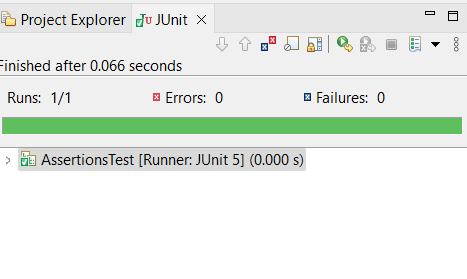
*assertSame*("JUnit", "JUnit");

*assertNotSame*("abc", new String("abc"));

*assertArrayEquals*(new int[]{1, 2, 3}, new int[]{1, 2, 3});

}

}



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in JUnit**

**BankAccount.java**

package testdemo3;

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

balance += amount;

}

public boolean withdraw(int amount) {

if (amount <= balance) {

balance -= amount;

return true;

}

return false;

}

public int getBalance() {

return balance;

}

}

**BankAccountTest.java**

package testdemo3;

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class BankAccountTest {

private BankAccount account;

@Before

public void setup() {

System.*out*.println("Setting up account");

account = new BankAccount(100);

}

@After

public void teardown() {

System.*out*.println("Tearing down account");

account = null;

}

@Test

public void testDeposit() {

int amount = 50;

account.deposit(amount);

*assertEquals*(150, account.getBalance());

}

@Test

public void testWithdrawSuccess() {

int amount = 40;

boolean success = account.withdraw(amount);

*assertTrue*(success);

*assertEquals*(60, account.getBalance());

}

@Test

public void testWithdrawFail() {

int amount = 200;

boolean success = account.withdraw(amount);

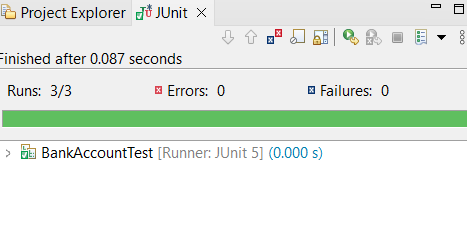
*assertFalse*(success);

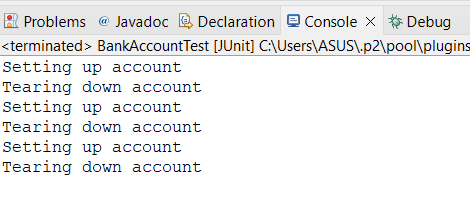
*assertEquals*(100, account.getBalance());

}

}

OUTPUT:





**MOCKITO**

**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.

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

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);

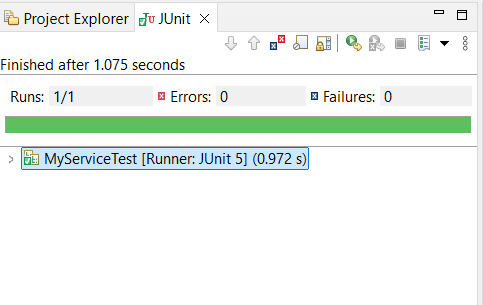
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

OUTPUT:



**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.

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example;

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:

