**JUnit Basic 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

3. Create a new test class in your project.

**IMPLEMENTATION :**

**Step1 :**

Created a Maven-based Java project named **JUnitDemo**.  
Added JUnit 4.13.2 dependency in pom.xml to enable unit testing support.

**Step2 :**

Added the JUnit dependency inside the <dependencies> section of the pom.xml file.

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Step3 :**

**PrimeChecker.java**

public class PrimeChecker {

public boolean isPrime(int number) {

if (number <= 1) return false;

for (int i = 2; i <= Math.sqrt(number); i++) {

if (number % i == 0) return false;

}

return true;

}

}

**PrimeCheckerTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class PrimeCheckerTest {

@Test

public void testPrimeNumber() {

assertTrue(new PrimeChecker().isPrime(11));

}

@Test

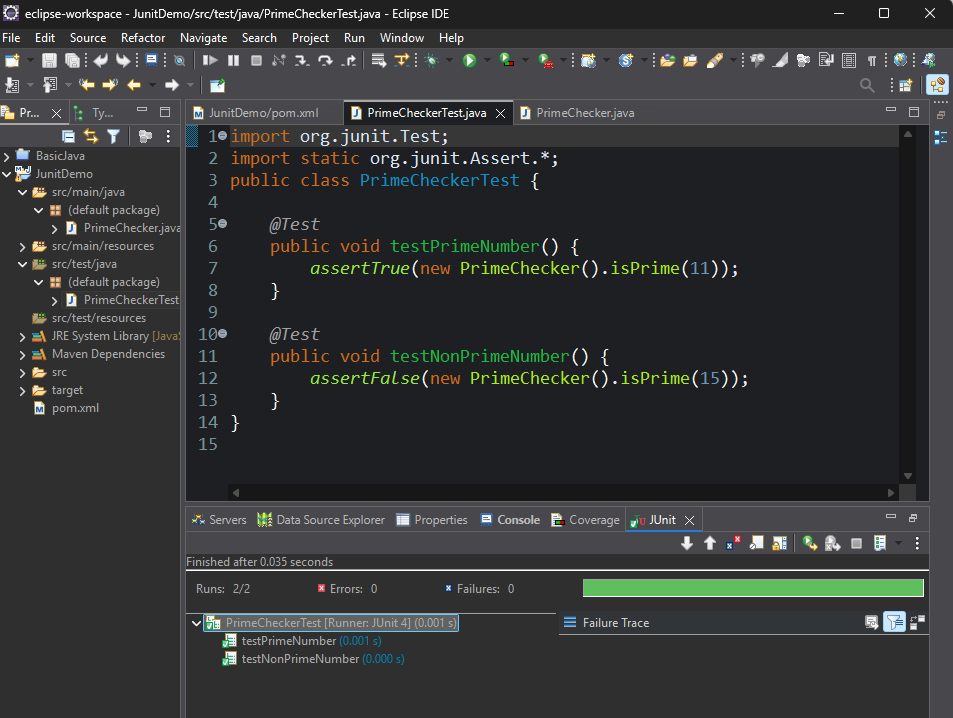
public void testNonPrimeNumber() {

assertFalse(new PrimeChecker().isPrime(15));

}

}

**OUTPUT SCREENSHOT:**



***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.
2. 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);

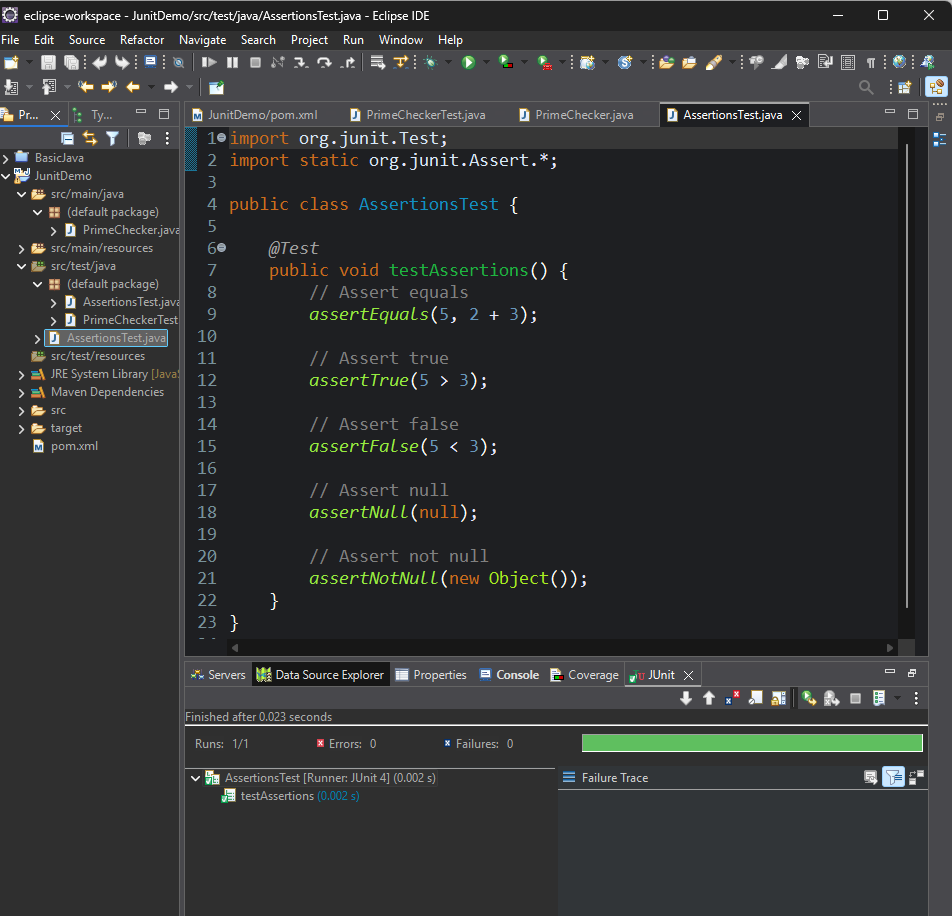
// Assert not null

assertNotNull(new Object());

}

}

**OUTPUT SCREENSHOT:**



***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.

**IMPLEMENTATION :**

**Class Under Test- BankAccount.java**

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

if (amount <= balance) {

balance -= amount;

}

}

public int getBalance() {

return balance;

}

}

**JUnit Test Class – BankAccountTest.java**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.*assertEquals*;

public class BankAccountTest {

private BankAccount account;

*@Before*

public void setUp() {

account = new BankAccount(1000);

System.***out***.println("Account setup with balance = " + account.getBalance());

}

*@After*

public void tearDown() {

account = null;

System.***out***.println("Teardown done.");

}

*@Test*

public void testDeposit() {

// Act

account.deposit(500);

// Assert

*assertEquals*(1500, account.getBalance());

}

*@Test*

public void testWithdraw() {

// Act

account.withdraw(300);

// Assert

*assertEquals*(700, account.getBalance());

}

*@Test*

public void testWithdrawMoreThanBalance() {

// Act

account.withdraw(1500); // Should not allow

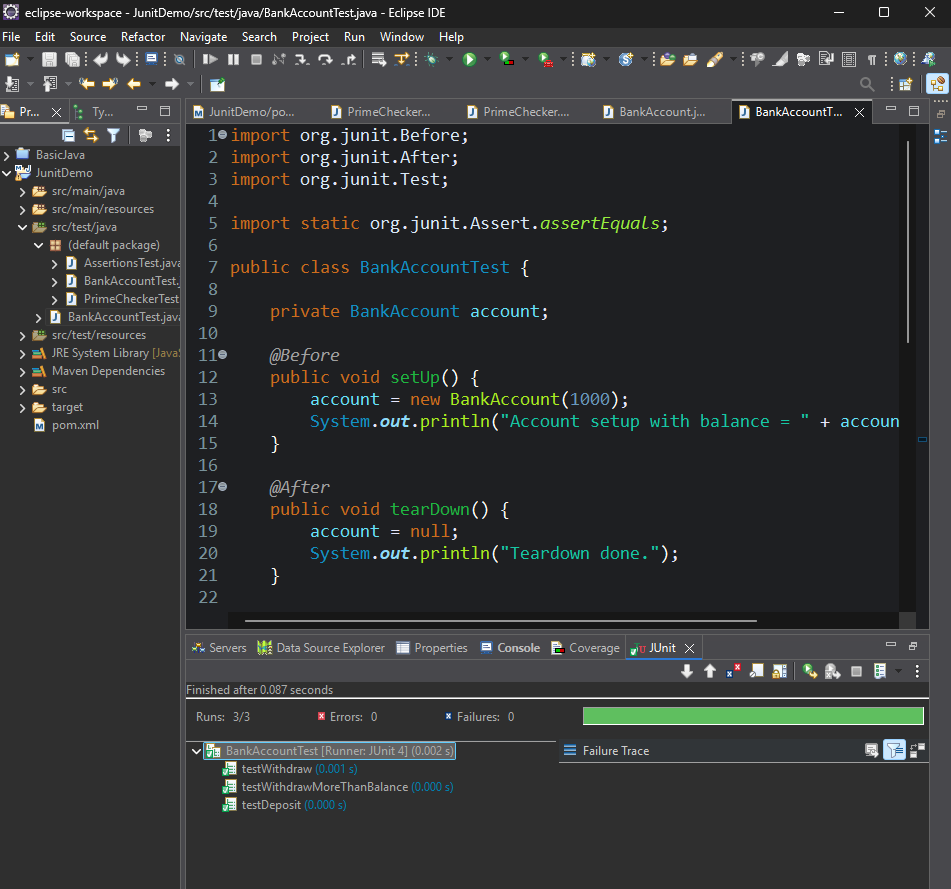
// Assert

*assertEquals*(1000, account.getBalance()); // Balance unchanged

}

}

**OUTPUT SCREENSHOT:**



**Mockito Hands-on Excercises**

***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.

**IMPLEMENTATION :**

**ExternalApi.java**

public interface ExternalApi {

String getData();

}

**MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

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);

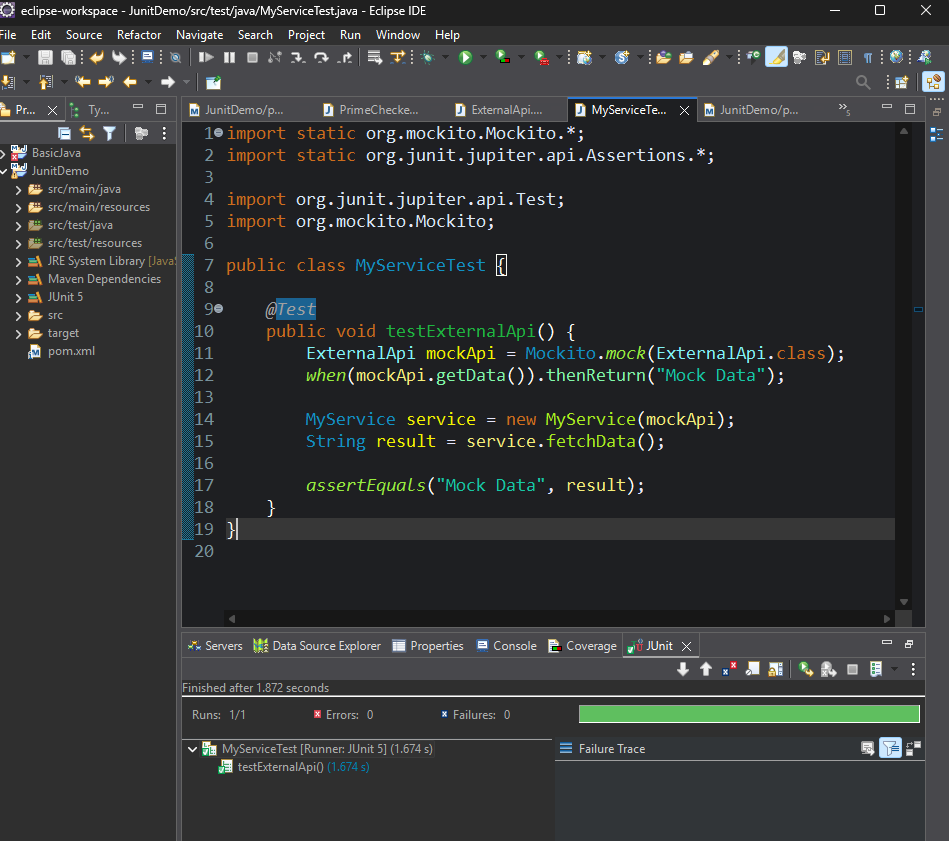
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT SCREENSHOT :**



***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.

**IMPLEMENTATION :**

**ExternalApi.java**

public interface ExternalApi {

String send(String message);

}

**MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.send("Test Message");

}

}

**MyServiceTest.java**

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.process();

*verify*(mockApi).send("Test Message");

}

}

**OUTPUT SCREENSHOT:**

