**JUNIT BASICS EXERCISE**

**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>

**Step 3:**

**StringUtils.java**

public class StringUtils {

public boolean isPalindrome(String input) {

if (input == null) return false;

String clean = input.replaceAll("\\s+", "").toLowerCase();

return clean.equals(new StringBuilder(clean).reverse().toString());

}

public String reverse(String input) {

if (input == null) return null;

return new StringBuilder(input).reverse().toString();

}

}

**StringUtilsTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

@Test

public void testIsPalindrome() {

StringUtils utils = new StringUtils();

*assertTrue*(utils.isPalindrome("madam"));

*assertTrue*(utils.isPalindrome("RaceCar"));

*assertFalse*(utils.isPalindrome("hello"));

*assertFalse*(utils.isPalindrome(null));

}

@Test

public void testReverse() {

StringUtils utils = new StringUtils();

*assertEquals*("cba", utils.reverse("abc"));

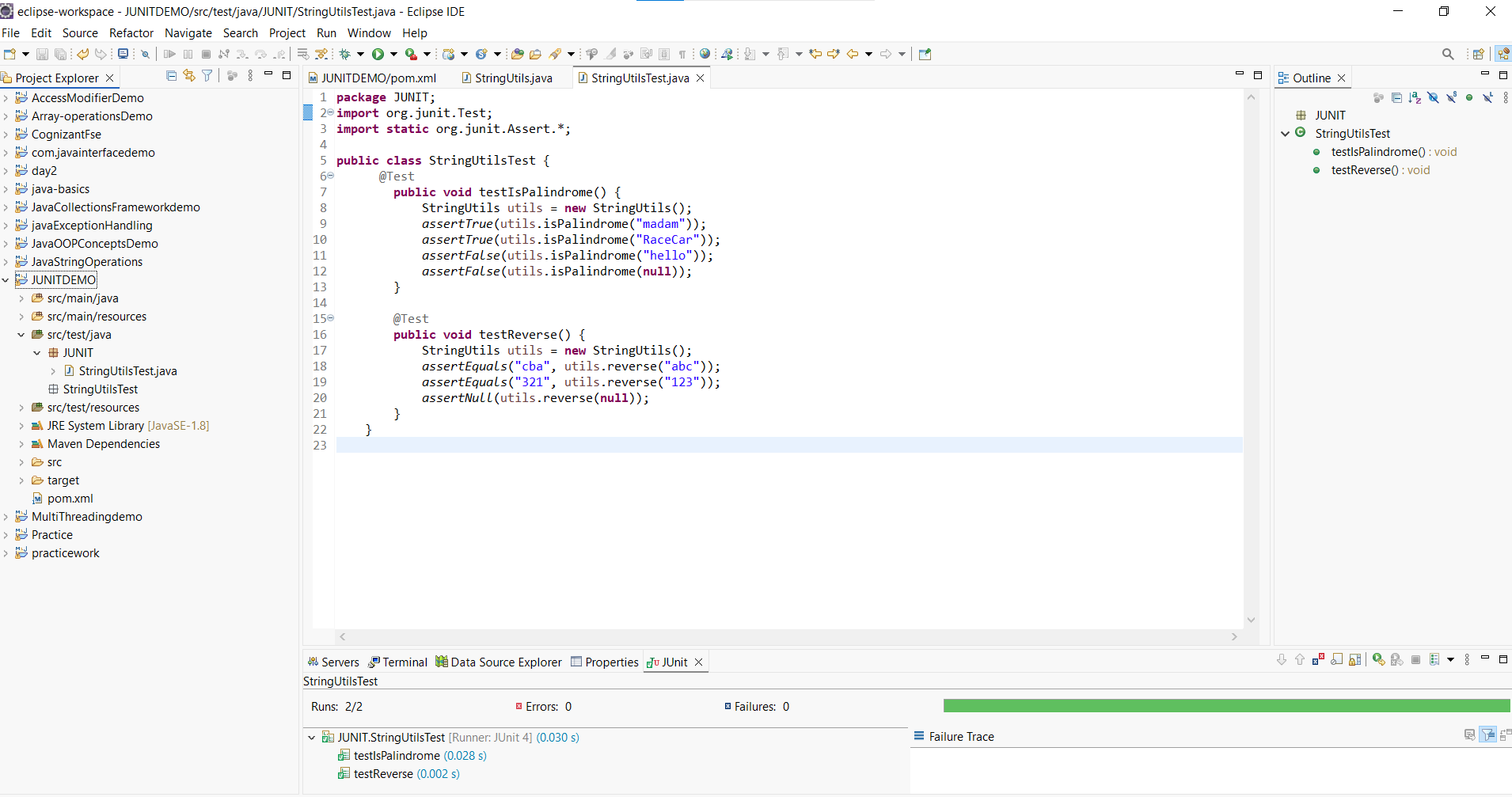
*assertEquals*("321", utils.reverse("123"));

*assertNull*(utils.reverse(null));

}

}

**OUTPUT:**

****

**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:**

package JUNIT;

import org.junit.Test;

import static org.junit.Assert.\*;

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);

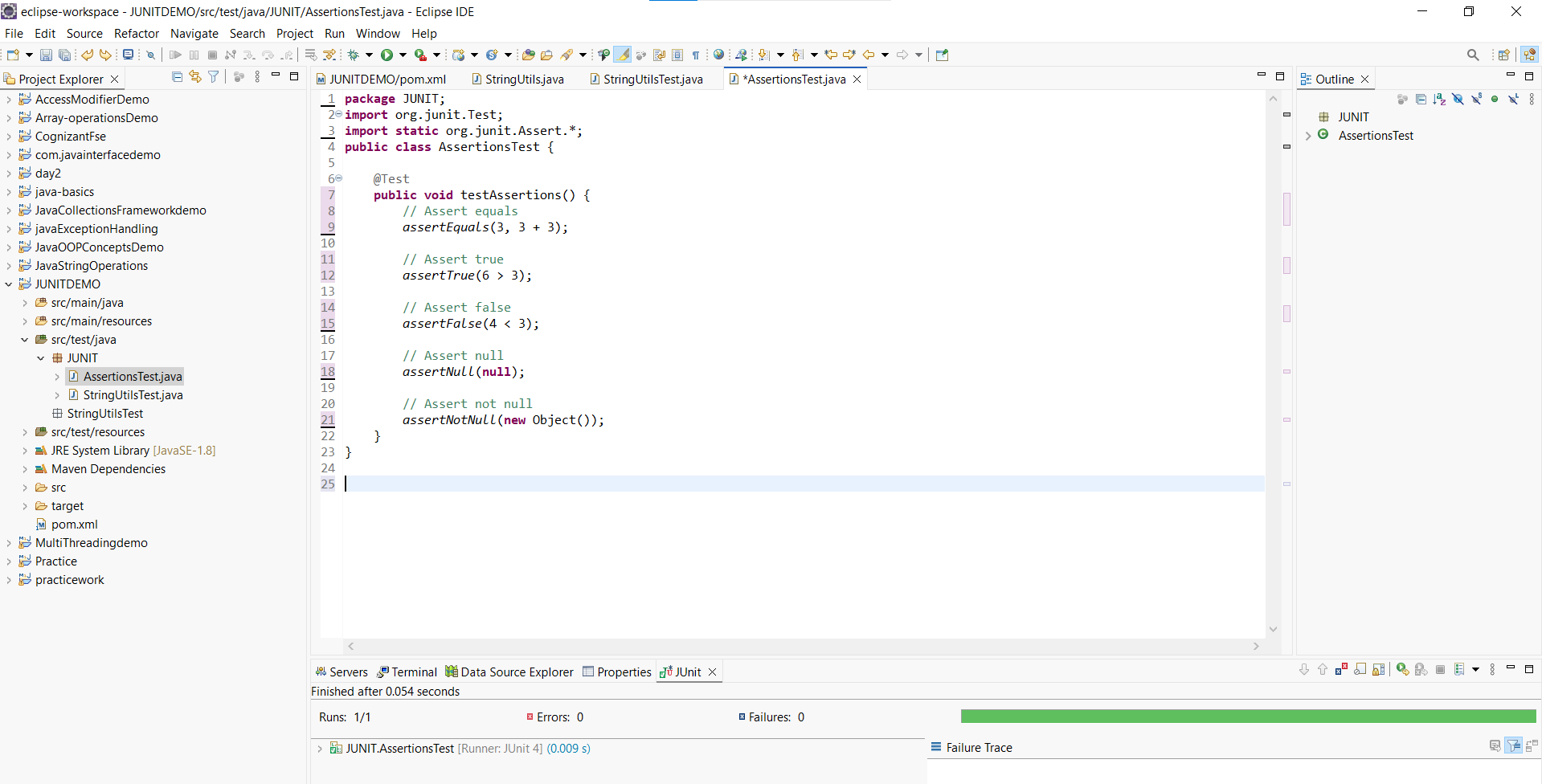
// Assert not null

assertNotNull(new Object());

}

}

**OUTPUT:**

****

***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 :**

**Temperature Converter.java:**

package JUNIT;

public class TemperatureConverter {

public double celsiusToFahrenheit(double celsius) {

return celsius \* 9 / 5 + 32;

}

public double fahrenheitToCelsius(double fahrenheit) {

return (fahrenheit - 32) \* 5 / 9;

}

}

**TemperatureConverterTest.java:**

package JUNIT;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class TemperatureConverterTest {

private TemperatureConverter converter;

@Before

public void setUp() {

converter = new TemperatureConverter(); // Arrange

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

}

@After

public void tearDown() {

converter = null;

System.out.println("Teardown complete");

}

@Test

public void testCelsiusToFahrenheit() {

// Act

double result = converter.celsiusToFahrenheit(0);

// Assert

assertEquals(32.0, result, 0.01);

}

@Test

public void testFahrenheitToCelsius() {

// Act

double result = converter.fahrenheitToCelsius(212);

// Assert

assertEquals(100.0, result, 0.01);

}

@Test

public void testFreezingPoint() {

// Act

double result = converter.celsiusToFahrenheit(-40);

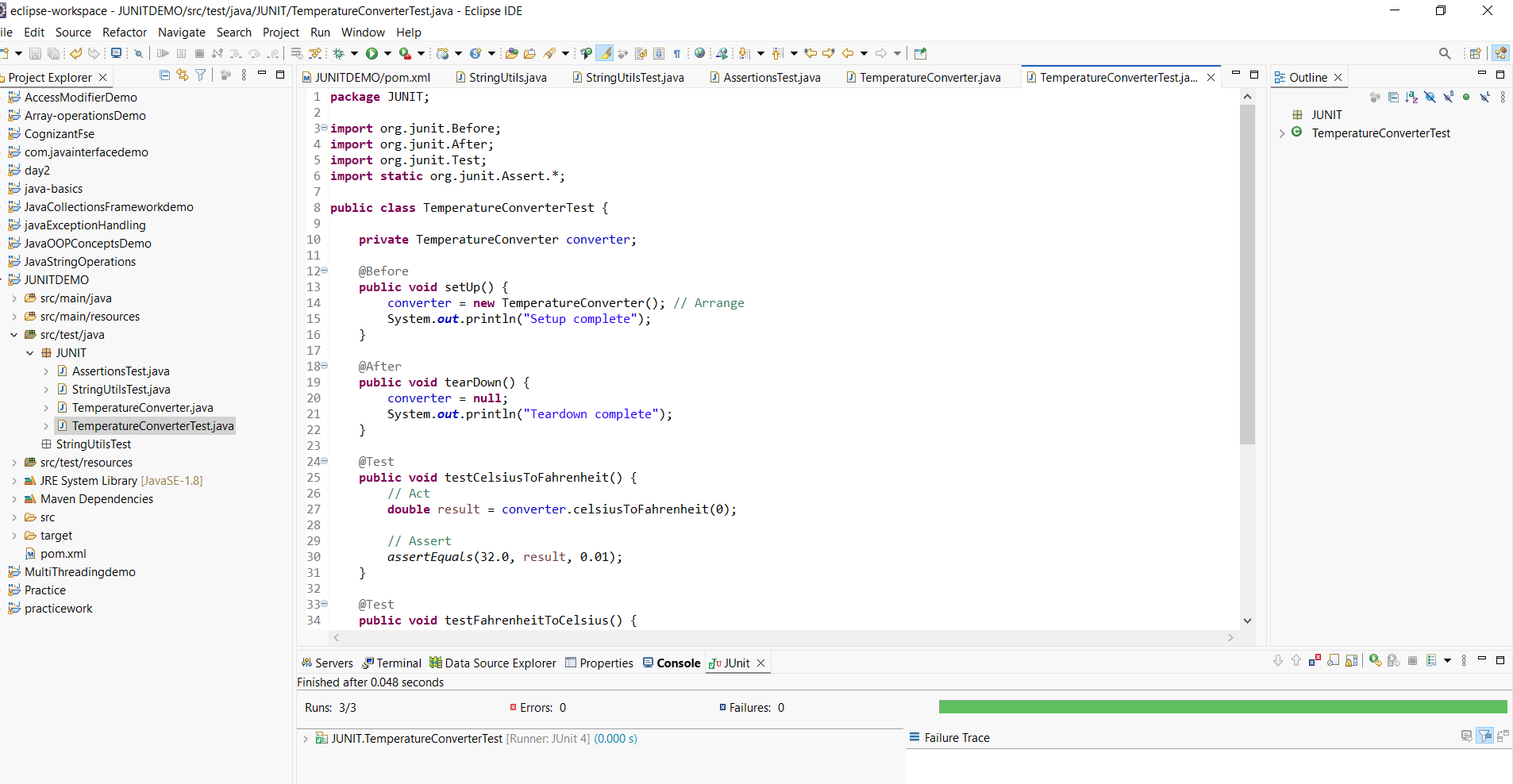
// Assert

assertEquals(-40.0, result, 0.01); // special case: -40°C = -40°F

}

}

**OUTPUT:**

****

**MOCKITO EXRECISES**

**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);

***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:

