**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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

**Step 1: Creating a Maven Project**

* Open Eclipse IDE.
* Go to: File > New > Maven Project
* Choose the archetype: maven-archetype-quickstart
* Enter project details:
  + **Group Id**: com.example
  + **Artifact Id**: JUnitDemo1
  + **Version**: 0.0.1-SNAPSHOT
  + **Package**: com.example.JUnitDemo1
* Finish and let Maven download dependencies.

**Step 2: Verification Project Creation**

* In **Project Explorer**,

JUnitDemo1

* + src/main/java
  + src/test/java

**Step 3: Sample Junit Example\_ Multiplication**

Testing a utility class that converts:

* MathUtil

**Step 4: Add a Java Class to Test**

* Navigate to src/main/java/com.example.JUnitDemo1
* Create a class named MathUtilTest

**MathUtil.java:**

package JUnitDemo1;

public class MathUtil

{

private int a;

private int b;

public int multiply (int a,int b) {

return 0;

}

}

**MathUtilTest.java under src/test/java**

package JUnitDemo1;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MathUtilTest {

@Test

public void testMultiply() {

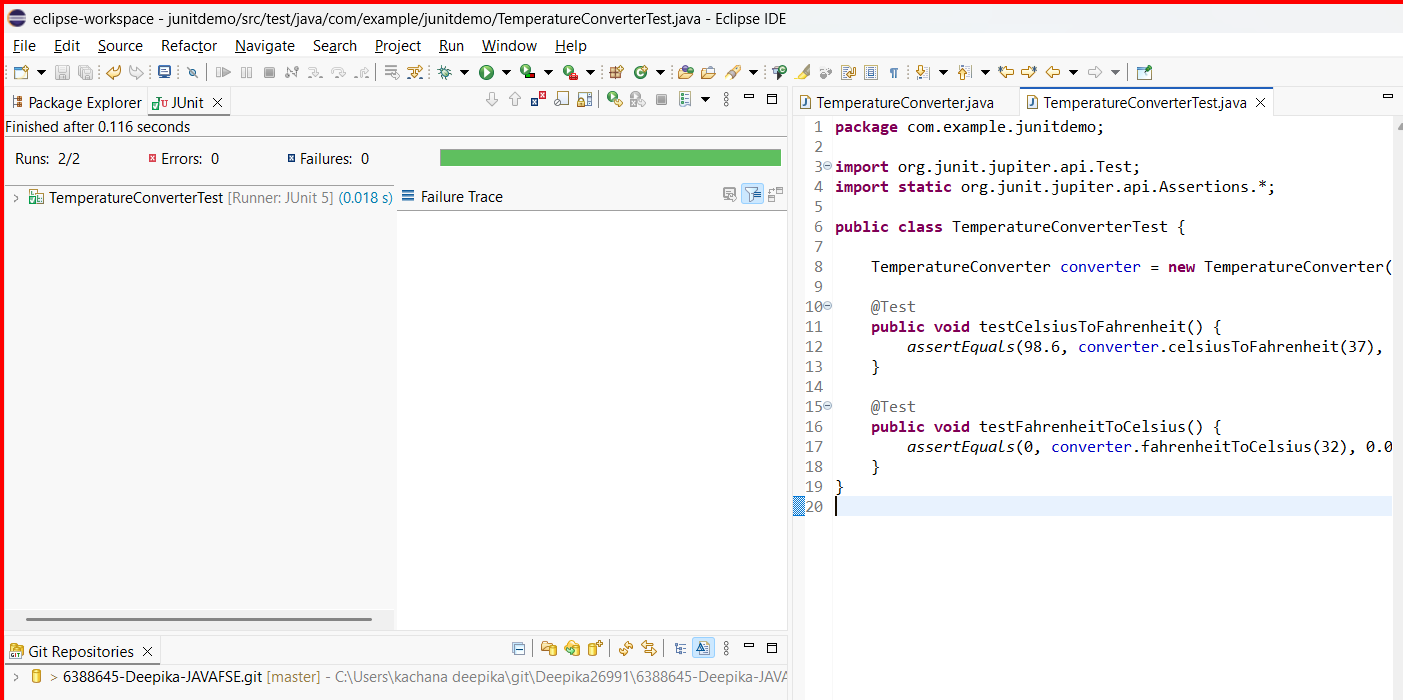
MathUtil util = new MathUtil();

assertEquals(24,util.multiply(4, 6));

}

}

**Output:**

****

**Exercise 3:**

**Assertions in JUnit Scenario:**

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

**Assertion in Unit Testing:**

An assertion is a statement in a unit test that checks whether a specific condition is true.  
If the condition is true, the test passes.  
If the condition is false, the test fails, indicating something is wrong in your code.

**Purpose of Assertions:**

* To **validate expected outcomes** of code.
* To **automatically detect bugs**.
* To ensure the **correctness** of your logic.

Variety of assertions provided by JUnit, such as:

* assertEquals
* assertTrue
* assertFalse
* assertNull
* assertNotNull

**AssertionsTest.java:**

package com.example.junitdemo;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

// assertEquals: Checks if two values are equal

assertEquals(5, 2 + 3, "Expected 2 + 3 to equal 5");

// assertTrue: Asserts that the condition is true

assertTrue(5 > 3, "Expected condition to be true");

// assertFalse: Asserts that the condition is false

assertFalse(5 < 3, "Expected condition to be false");

// assertNull: Asserts that the object is null

assertNull(null, "Expected value to be null");

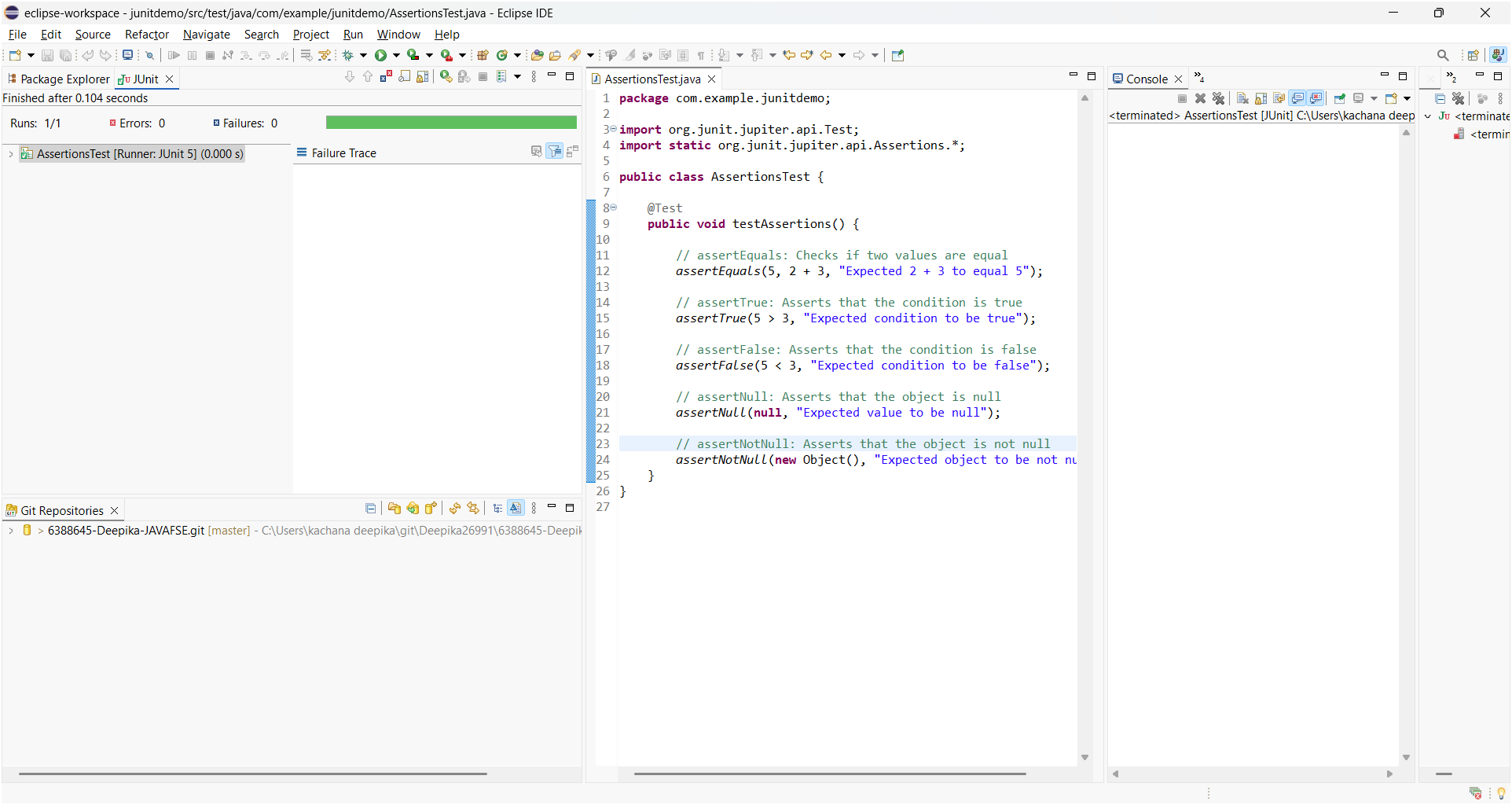
// assertNotNull: Asserts that the object is not null

assertNotNull(new Object(), "Expected object to be not null");

}

}

**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.

The AAA Pattern stands for:

**1. Arrange:**

Prepare everything required for the test:

* Create objects
* Set initial values
* Define input data
* Example : BankAccount account = new BankAccount(1000);

**2. Act:**

* Execute the method or function you want to test.
* Example: account.deposit(500)

**3. Assert:**

* Check if the actual result matches the expected result.
* Example: assertEquals(1500, account.getBalance());

It is a structured approach used in unit testing to write clear, readable, and maintainable test cases.

**Benefits of AAA Pattern:**

* Improves readability & Clarifies test flow
* Helps isolate and debug errors quickly
* Makes tests easier to review and maintain

**JUnit Annotations for Setup and Teardown:**

| Annotation | Purpose |
| --- | --- |
| @BeforeEach | Runs before each @Test method |
| @AfterEach | Runs after each @Test method |

PasswordValidator.java:

package JUnitDemo1;

public class PasswordValidator {

public boolean isValid(String password) {

if (password == null || password.length() < 8) {

return false;

}

if (!password.matches(".[@#$%^&+=].")) {

return false;

}

if (password.contains(" ")) {

return false;

}

return true;

}

}

PasswordValidator.java:

package JUnitDemo1;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

private PasswordValidator validator;

@Before

public void setUp() {

// Arrange

validator = new PasswordValidator();

}

@After

public void tearDown() {

validator = null;

}

@Test

public void testValidPassword() {

// Act

boolean result = validator.isValid("Secure@123");

// Assert

assertTrue(result);

}

@Test

public void testPasswordTooShort() {

// Act

boolean result = validator.isValid("S@1");

// Assert

assertFalse(result);

}

@Test

public void testPasswordWithoutSpecialCharacter() {

// Act

boolean result = validator.isValid("Password123");

// Assert

assertFalse(result);

}

@Test

public void testPasswordWithSpaces() {

// Act

boolean result = validator.isValid("Secu re@1");

// Assert

assertFalse(result);

}

@Test

public void testNullPassword() {

// Act

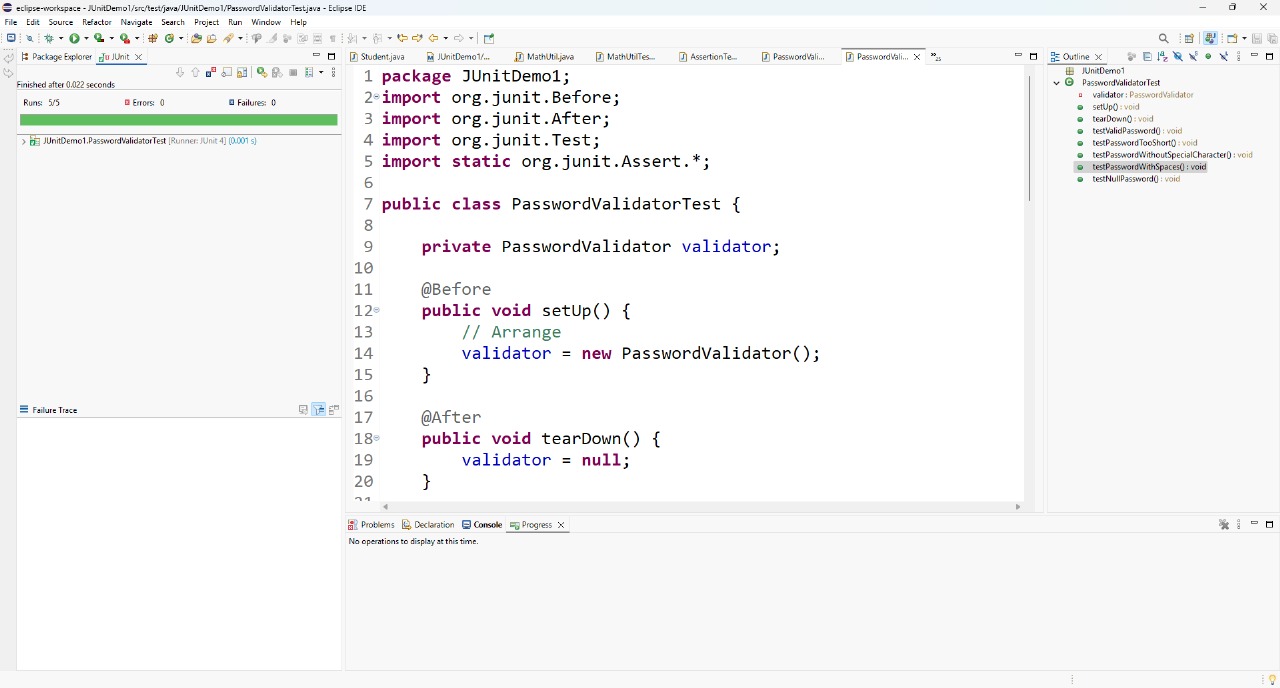
boolean result = validator.isValid(null);

// Assert

assertFalse(result);

}

}**Output:**

****