**Week 2 \_TDD using JUnit5**

**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**: junitdemo
  + **Version**: 0.0.1-SNAPSHOT
  + **Package**: com.example.junitdemo
* Finish and let Maven download dependencies.

**Step 2: Verification Project Creation**

* In **Project Explorer**,

Junitdemo

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

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

Testing a utility class that converts:

* Celsius to Fahrenheit
* Fahrenheit to Celsius

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

* Navigate to src/main/java/com.example.junitdemo
* Create a class named TemperatureConverter

**TemperatureConverter.java:**

package com.example.junitdemo;

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 under src/test/java**

package com.example.junitdemo;

import org.junit.jupiter.api.Test;

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

public class TemperatureConverterTest {

TemperatureConverter converter = new TemperatureConverter();

@Test

public void testCelsiusToFahrenheit() {

assertEquals(98.6, converter.celsiusToFahrenheit(37), 0.001);

}

@Test

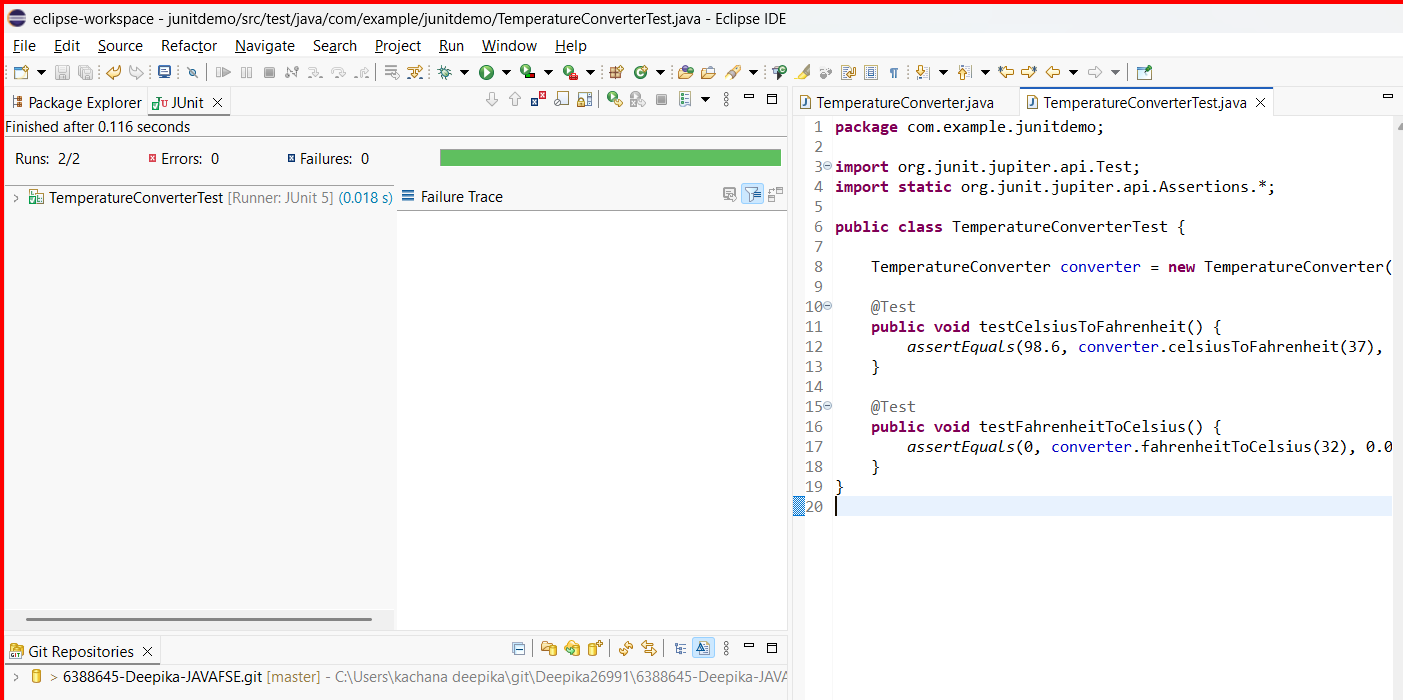
public void testFahrenheitToCelsius() {

assertEquals(0, converter.fahrenheitToCelsius(32), 0.001);

}

}

**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");

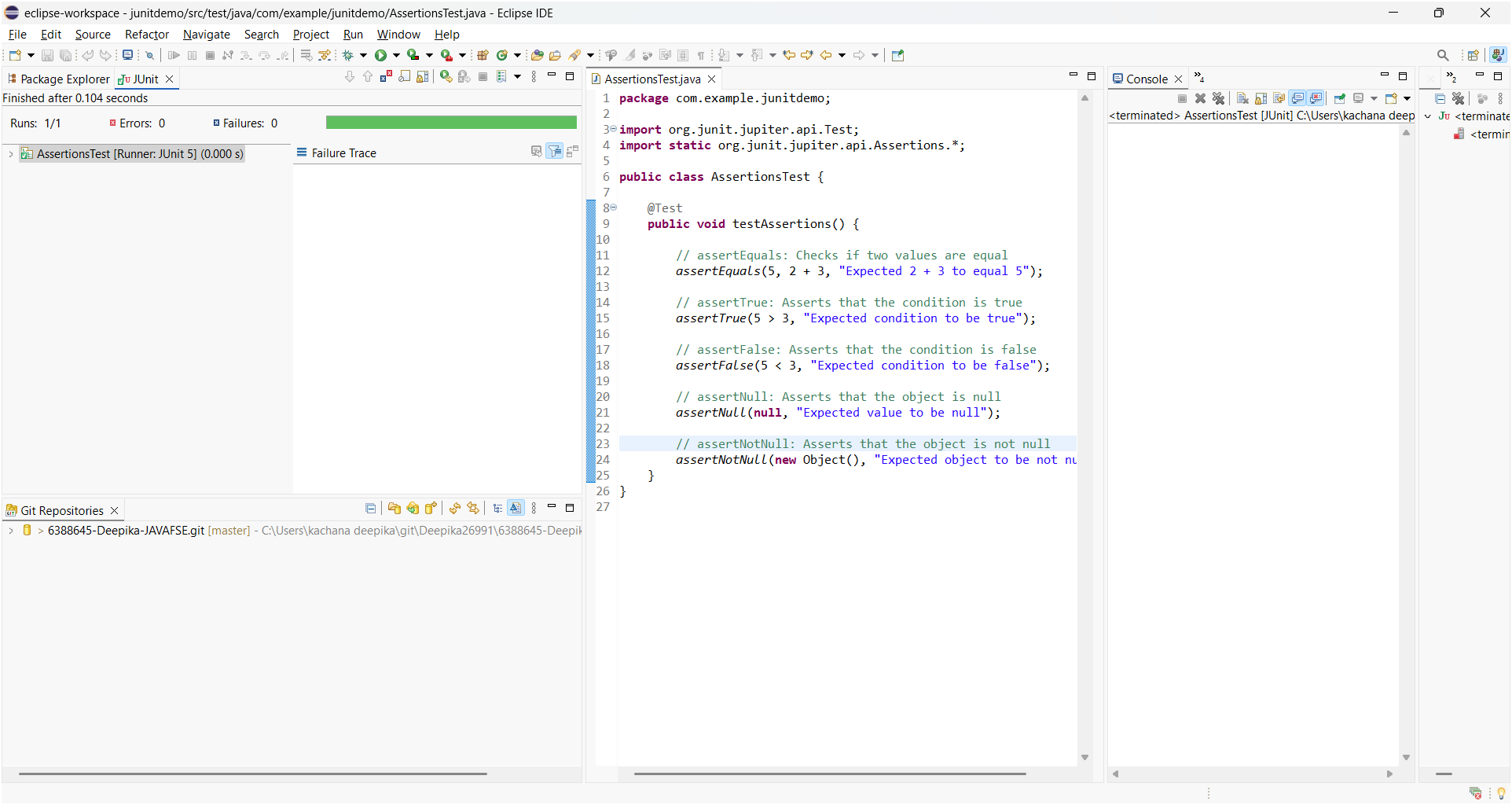
// assertNotNull: Asserts that the object is not null

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 |

BankAccount.java:

package com.example.junitdemo;

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

balance = initialBalance;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

}

BankAccountTest.java:

package com.example.junitdemo;

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

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

public class BankAccountTest

{

BankAccount account;

@BeforeEach

public void setUp()

{

// Arrange: create a fresh account before each test

account = new BankAccount(1000.0);

System.out.println("Account created with balance 1000");

}

@AfterEach

public void tearDown()

{

// Clean-up (not required here, but for demo)

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

}

@Test

public void testDeposit()

{

// Act

account.deposit(500.0);

// Assert

assertEquals(1500.0, account.getBalance());

}

@Test

public void testWithdraw()

{

// Act

account.withdraw(300.0);

// Assert

assertEquals(700.0, account.getBalance());

}

@Test

public void testWithdrawTooMuch() {

// Act

account.withdraw(1500.0); // More than balance

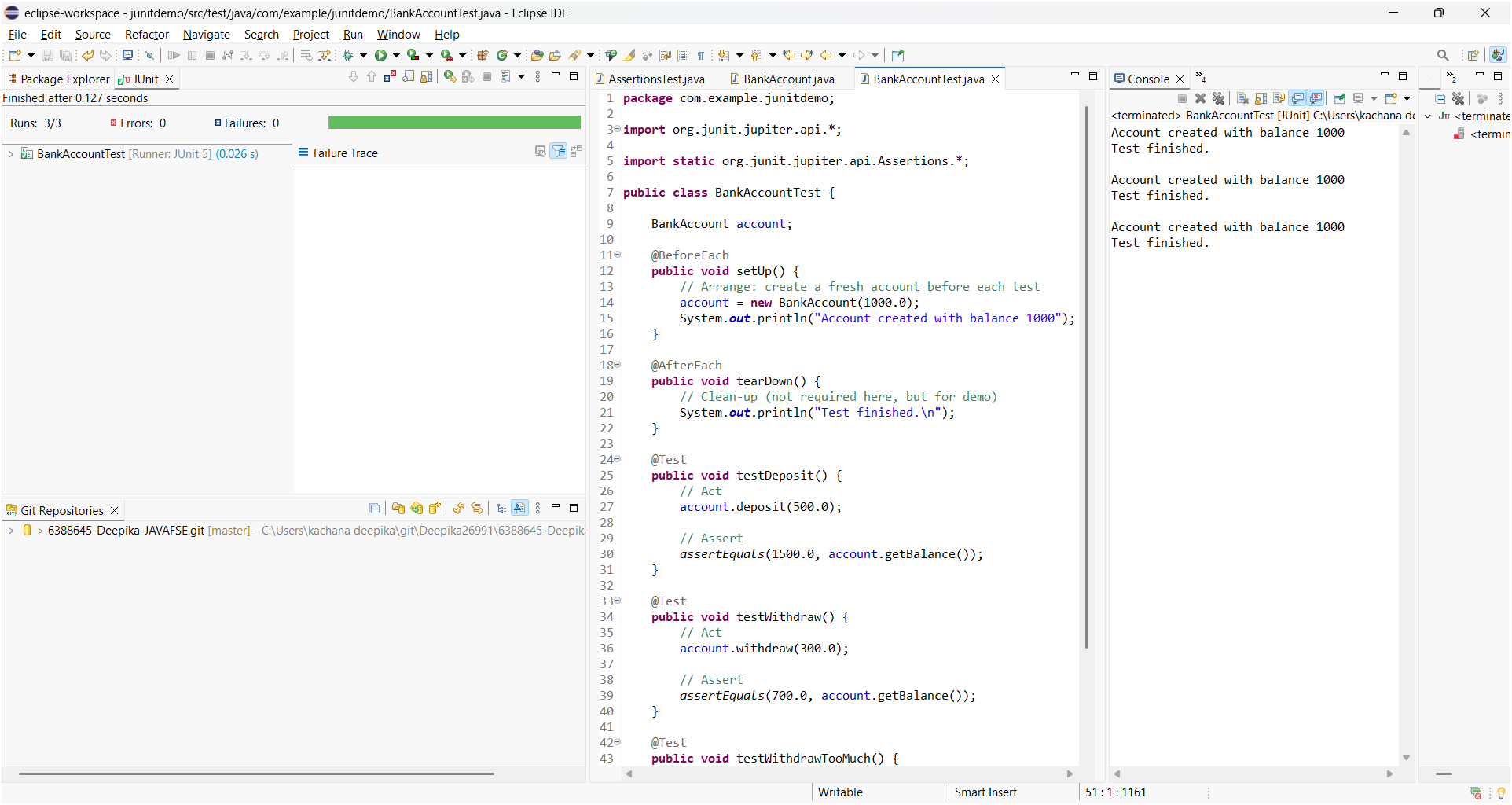
// Assert: Balance should remain unchanged

assertEquals(1000.0, account.getBalance());

}

}

**Output:**

****