**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests**

**1.Addition.java**

package com.example;

public class Addition {

public int add(int a, int b) {

return a + b;

}

}

**2.AdditionTest.java**

package com.example;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class AdditionTest {

@Test

public void testAdd() {

System.***out***.println("Calculation Started.....");

System.***out***.println("Addition of two number started.....");

Addition Add = new Addition();

int result = calc.add(2, 3);

assertEquals(5, result);

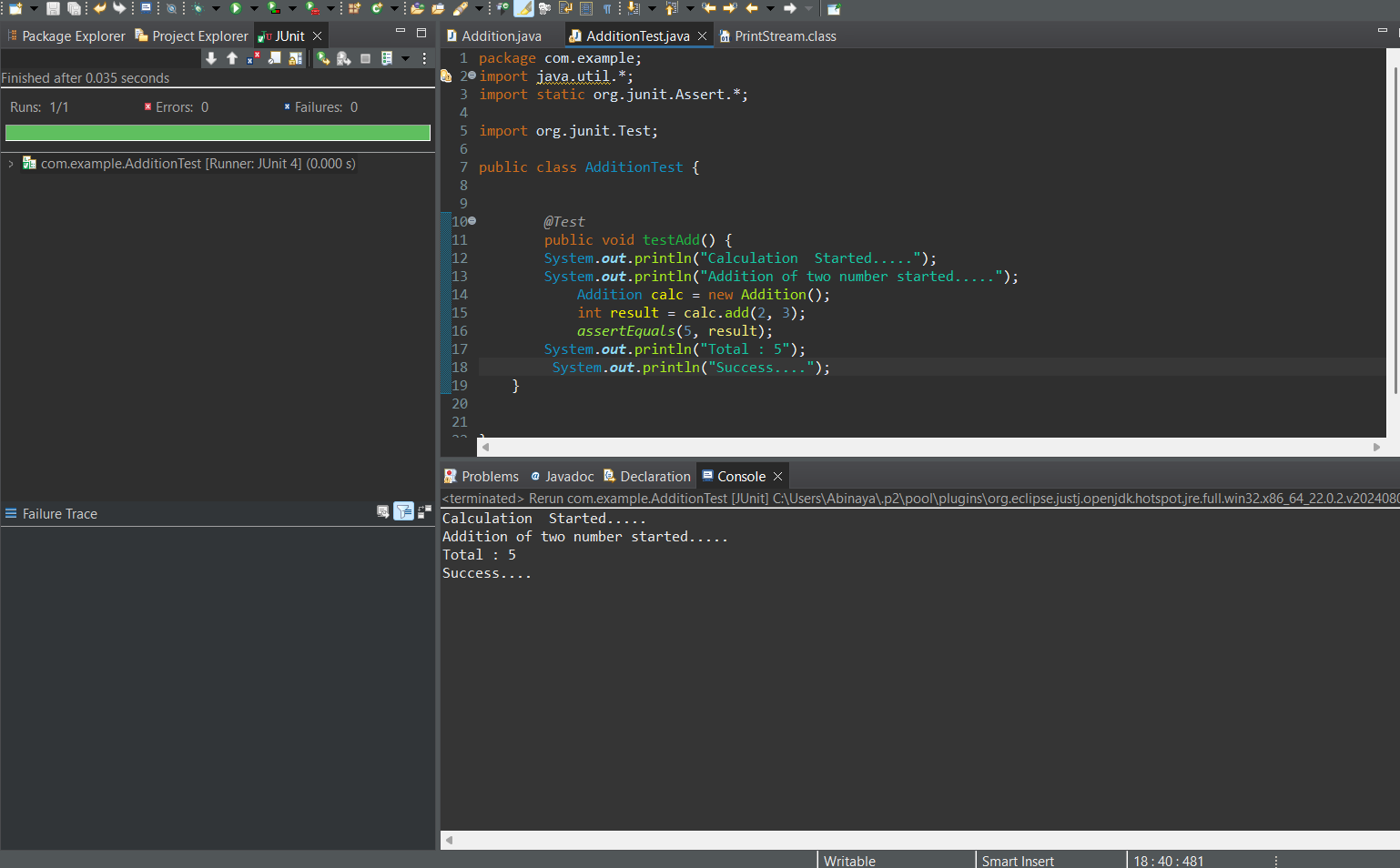
System.***out***.println("Total : 5");

System.***out***.println("Success....");

}

}

**Output:**



**Exercise 2: Writing Basic JUnit Tests Scenario: You need to write basic JUnit tests for a simple Java class.**

**1.MathUtils.java**

package com.example;

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**2.MathUtilsTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

MathUtils math = new MathUtils();

@Test

public void testAdd() {

assertEquals(5, math.add(2, 3));

assertEquals(0, math.add(-2, 2));

}

@Test

public void testSubtract() {

assertEquals(1, math.subtract(5, 4));

assertEquals(-3, math.subtract(2, 5));

}

@Test

public void testMultiply() {

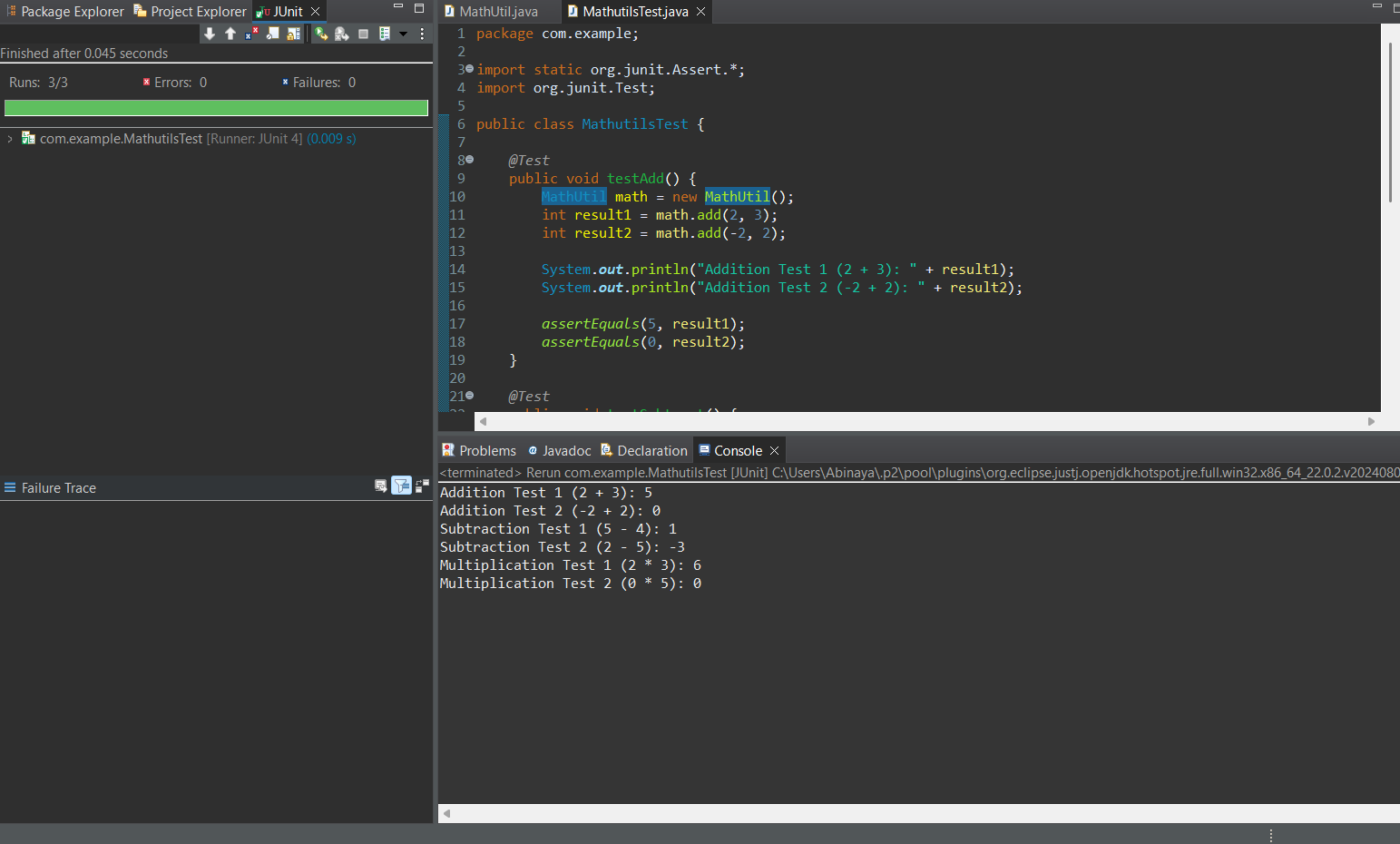
assertEquals(6, math.multiply(2, 3));

assertEquals(0, math.multiply(0, 5));

}

}

**Output:**



**Exercise 3: Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results.**

**1.User.java**

package com.example;

public class User {

private String name;

private int age;

private boolean active;

public User(String name, int age, boolean active) {

this.name = name;

this.age = age;

this.active = active;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public boolean isActive() {

return active;

}

}

**2.UserTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class UserTest {

@Test

public void testUserProperties() {

User user = new User("Taehyung", 25, true);

assertEquals("Taehyung", user.getName());

assertTrue(user.getAge() > 18);

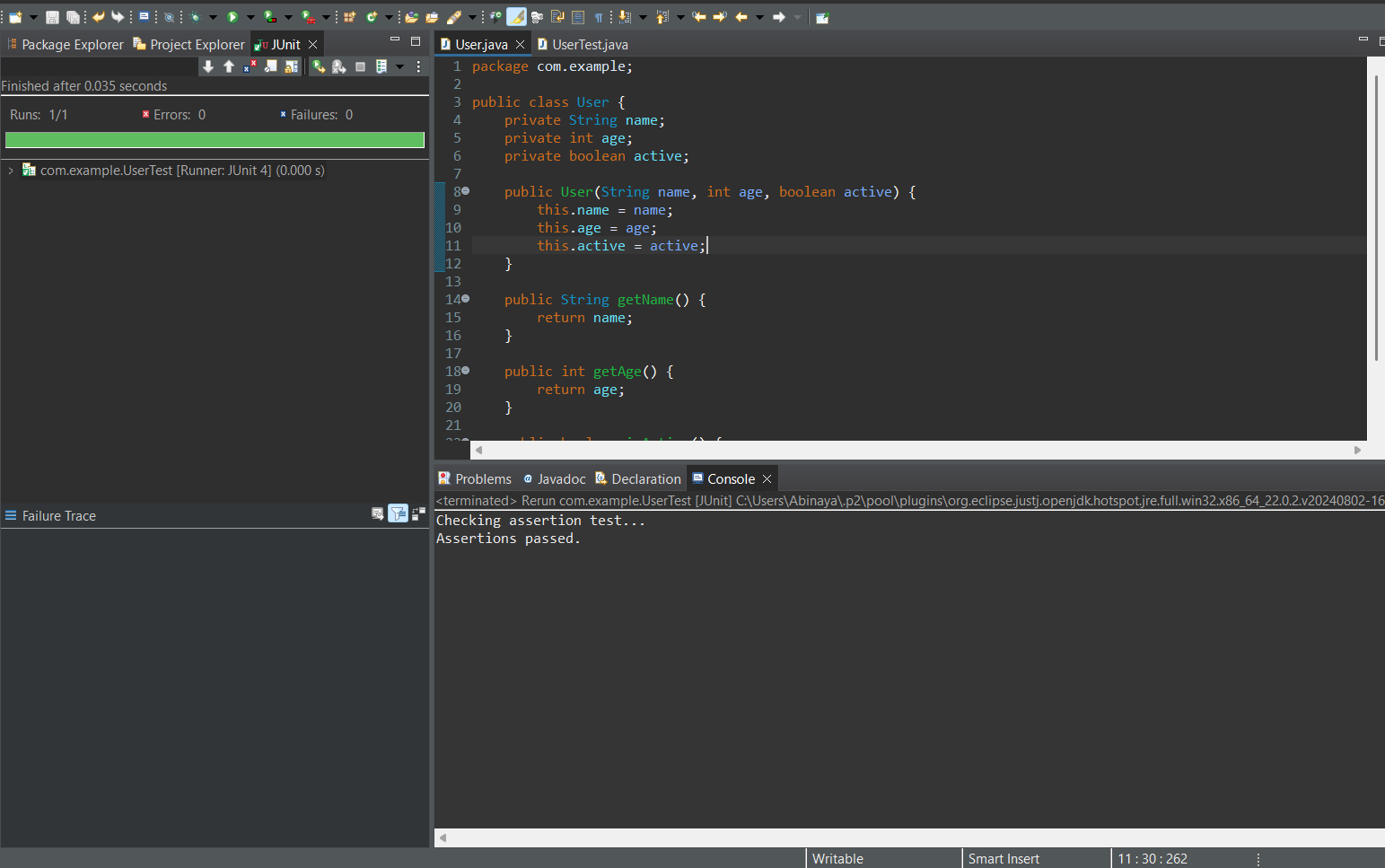
assertTrue(user.isActive());

assertNotNull(user.getName());

assertNotEquals(30, user.getAge());

}}

**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.**

**1.BankAccount.java**

package com.example;

public class BankAccount {

private String holderName;

private double balance;

public BankAccount(String holderName, double initialBalance) {

this.holderName = holderName;

this.balance = initialBalance;

}

public void deposit(double amount) {

balance += amount;

}

public boolean withdraw(double amount) {

if (amount > balance) return false;

balance -= amount;

return true;

}

public double getBalance() {

return balance;

}

public String getHolderName() {

return holderName;

}

}

**2.BankAccountTest.java**

package com.example;

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

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

public class BankAccountTest {

private BankAccount account;

@BeforeEach

public void setUp() {

account = new BankAccount("Abinaya", 1000.0); }

@AfterEach

public void tearDown() {

account = null;

}

@Test

public void testDeposit() {

// Arrange: Already done in setUp()

account.deposit(500);

assertEquals(1500.0, account.getBalance());}

@Test

public void testWithdrawWithSufficientBalance() {

boolean result = account.withdraw(400);

assertTrue(result);

assertEquals(600.0, account.getBalance());}

@Test

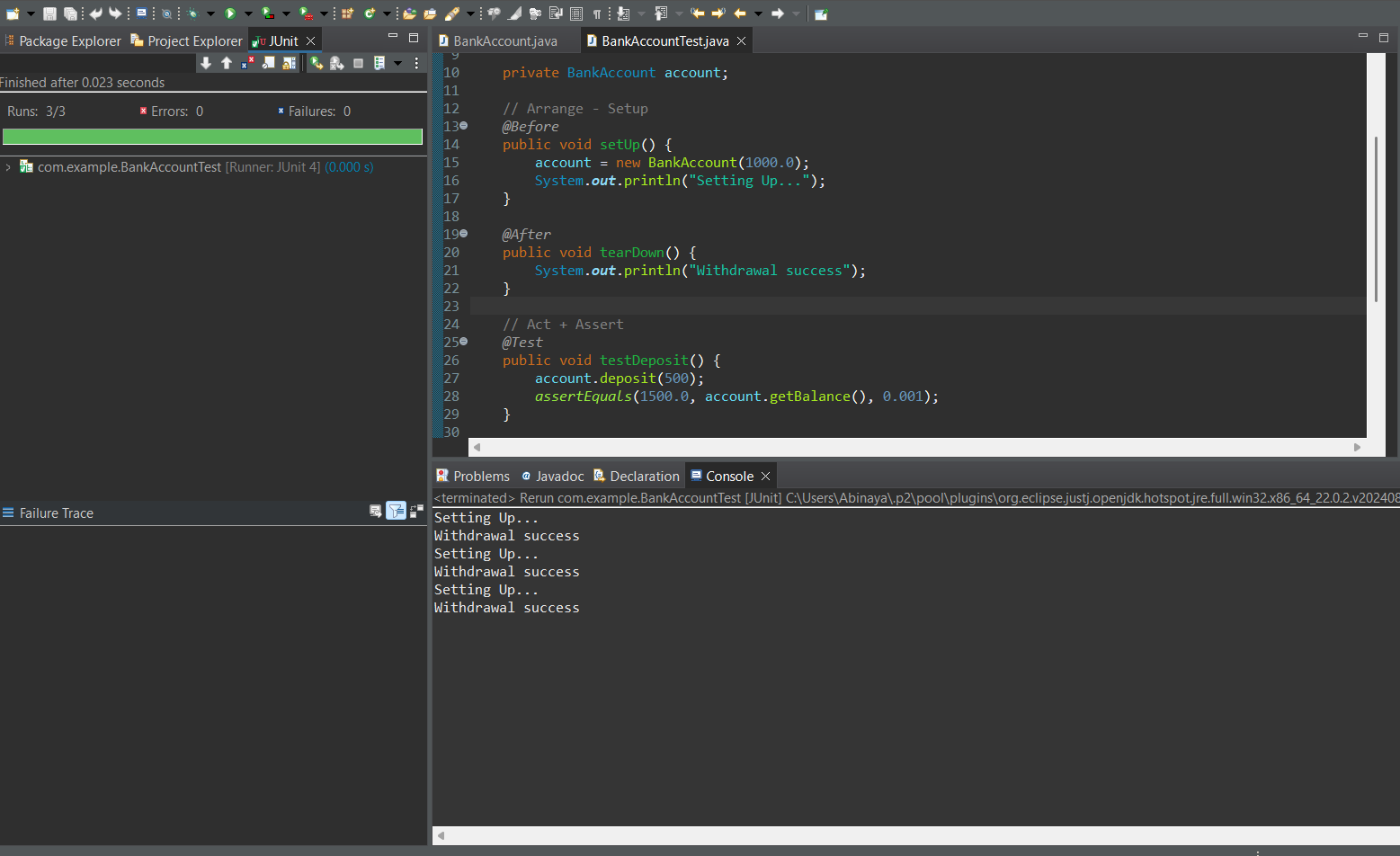
public void testWithdrawWithInsufficientBalance() {

boolean result = account.withdraw(1200);

assertFalse(result);

assertEquals(1000.0, account.getBalance()); // No deduction

}}

**Output:**