**Exercise 1: Setting Up Junit**

Directory Set-up:

deepskilling/

├── pom.xml

└── src/

└── test/

└── java/

└── CalcTest.java

Code for CalcTest.java:

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

import org.junit.jupiter.api.Test;

public class CalcTest {

    @Test

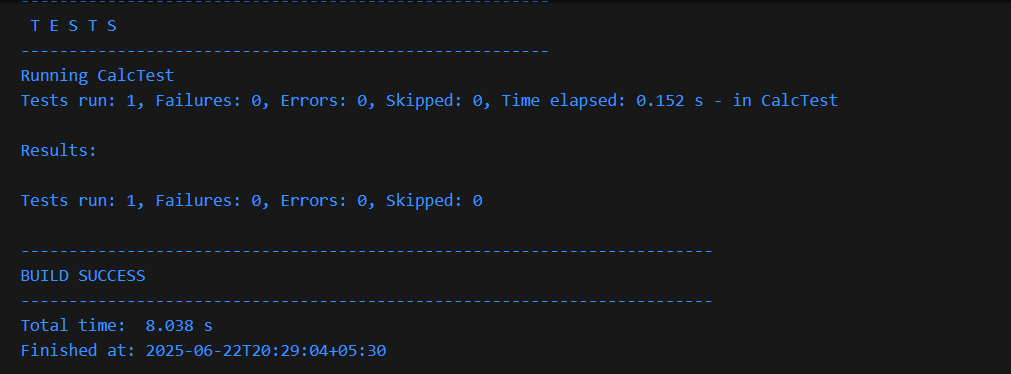
    public void test(){

        assertEquals(5,2+3);

    }

}

Ouput:



**Exercise 2: Writing Basic JUnit Tests**

Directory c:\Users\shreya ghosh\Downloads\deepskilling\

├── pom.xml

└── src\

├── main\

│ └── java\

│ └── Calculator.java

└── test\

└── java\

└── CalcTest.java

**Code for Calculator.java:**

public class Calculator {

    public int testAddition(int a, int b) {

        return a + b;

    }

    public int testSubtraction(int a, int b) {

        return a - b;

    }

    public int testMultiplication(int a, int b) {

        return a \* b;

    }

}

**Code for CalcTest:**

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

import org.junit.jupiter.api.Test;

public class CalcTest {

       Calculator calculator = new Calculator();

    @Test

    public void testAdd(){

        assertEquals(5, calculator.testAddition(2, 3));

    }

    @Test

    public void testSubtract(){

        assertEquals(1, calculator.testSubtraction(3, 2));

    }

    @Test

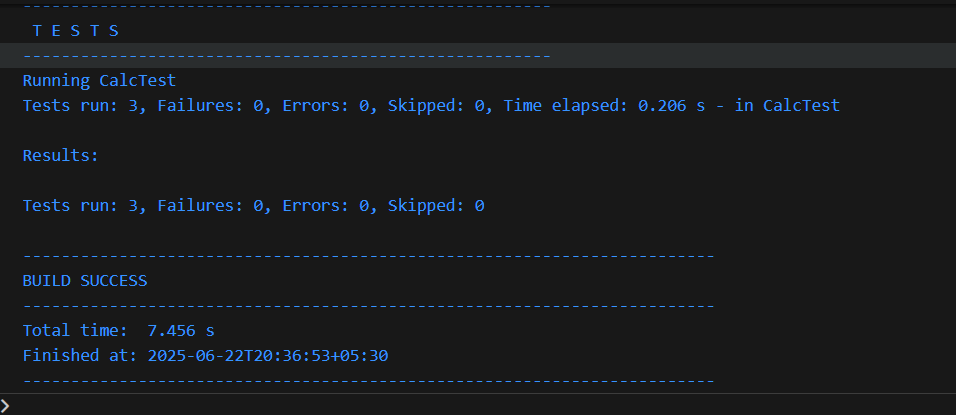
    public void testMultiply(){

        assertEquals(6, calculator.testMultiplication(2, 3));

    }

}

**Output:**



**Exercise 3: Assertions in Junit:**

Directories: :\Users\shreya ghosh\Downloads\deepskilling\

├── pom.xml

└── src\

├── main\

└── test\

└── java\

└── AssertTest.java

**Code:**

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

import org.junit.jupiter.api.Test;

 public class AssertTest {

    @Test

    public void  testassertequals(){

            assertEquals(2, 1 + 1);

    }

    @Test

    public void testassertnotequals(){

            assertNotEquals(2,5+3);

    }

    @Test

    public void testassertTrue(){

            assertTrue(5 > 3);

    }

    @Test

    public void testassertFalse(){

            assertFalse(3 > 5);

    }

        @Test

            public void testassertNull(){

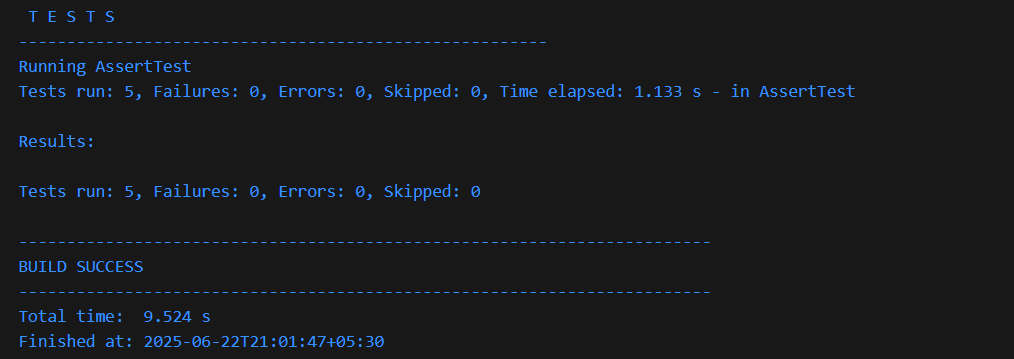
            String str = null;

            assertNull(str);

}

 }

**Output:**



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in Junit**

Directory c:\Users\shreya ghosh\Downloads\deepskilling\

├── pom.xml

└── src\

├── main\

│ └── java\

│ └── Bank.java

└── test\

└── java\

└── BankTest.java

**Code for Bank.java:**

public class Bank {

    double balance;

    public Bank(double initialBalance) {

        if (initialBalance < 0) {

            throw new IllegalArgumentException("Initial balance cannot be negative");

        }

        this.balance = initialBalance;

    }

    public double getBalance() {

        return balance;

    }

    public void deposit(double amount) {

        if (amount <= 0) {

            throw new IllegalArgumentException("Deposit amount must be positive");

        }

        balance += amount;

    }

    public void withdraw(double amount) {

        if (amount <= 0) {

            throw new IllegalArgumentException("Withdrawal amount must be positive");

        }

        if (amount > balance) {

            throw new IllegalArgumentException("Insufficient funds for withdrawal");

        }

        balance -= amount;

    }

}

**Code for BankTest.java:**

import org.junit.jupiter.api.Test;

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

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.BeforeEach;

public  class  BankTest {

    Bank bank;

    @BeforeEach

       void setUp() {

           bank = new Bank(100.0); //Arrange

    }

    @AfterEach

    void tearDown() {

        bank = null;

    }

    @Test

    public void testInitialBalance() {

        assertEquals(100.0,bank.getBalance());

    }

    @Test

      public void testnegativeBalance() {

        assertThrows(IllegalArgumentException.class, () -> {

            new Bank(-50.0);

        });

    }

    @Test

    public void testDeposit() {

        bank.deposit(50.0); //Act

        assertEquals(150.0, bank.getBalance()); //Assert

    }

    @Test

    public void testDepositNegativeAmount() {

        assertThrows(IllegalArgumentException.class, () -> {

            bank.deposit(-20.0);

        });

    }

    @Test

    public void testWithdraw() {

        bank.withdraw(30.0);

        assertEquals(70.0, bank.getBalance());

    }

    @Test

    public void testWithdrawNegativeAmount() {

        assertThrows(IllegalArgumentException.class, () -> {

            bank.withdraw(-10.0);

        });

    }

}

**Output:**

