JUnit Lab Assignment

Name   : **Abrar Khatib Lajim**

#ID   : **2022-3-60-043**

Course   : **CSE430**

Section   : **01**

Instructor : **Dr. Shamim H. Ripon**

Submission : **30/10/2025**

Lab Task 1

Source Code
TemperatureConverter.java:

```java
package Temperature;

public class TemperatureConverter {
    public double celsiusToFahrenheit(double c) {
        return (c * 9.0 / 5.0) + 32;
    }

    public double fahrenheitToCelsius(double f) {
        return (f - 32) * 5.0 / 9.0;
    }

    public double celsiusToKelvin(double c) {
        return c + 273.15;
    }
}
```

TemperatureConverterTest.java:

```java
package Temperature;

import org.junit.jupiter.api.Test;
```

```java
import static org.junit.jupiter.api.Assertions.*;

public class TemperatureConverterTest {

    private final TemperatureConverter converter = new TemperatureConverter();

    @Test
    void testCelsiusToFahrenheit() {
        assertEquals(32, converter.celsiusToFahrenheit(0), 0.01);
        assertEquals(212, converter.celsiusToFahrenheit(100), 0.01);
    }

    @Test
    void testFahrenheitToCelsius() {
        assertEquals(0, converter.fahrenheitToCelsius(32), 0.01);
        assertEquals(100, converter.fahrenheitToCelsius(212), 0.01);
    }

    @Test
    void testRoundTripConversion() {
        double c = 45.5;
        assertEquals(c, converter.fahrenheitToCelsius(converter.celsiusToFahrenheit(c)), 0.01);
    }

    @Test
    void testCelsiusToKelvin() {
        assertEquals(273.15, converter.celsiusToKelvin(0), 0.01);
    }
}
```
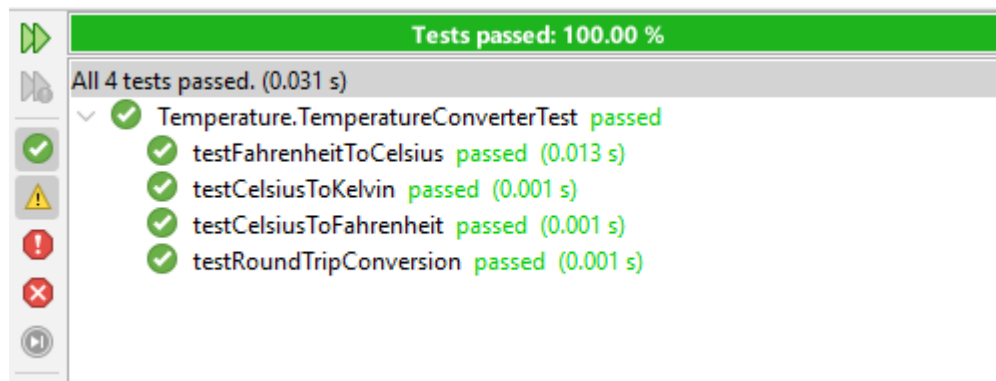
OutPut

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC1.1	celsiusToFahrenheit	0 °C	32 °F	Pass
TC1.2	celsiusToFahrenheit	100 °C	212 °F	Pass
TC1.3	fahrenheitToCelsius	212 °F	100 °C	Pass
TC1.4	celsiusToKelvin	25 °C	298.15 K	Pass
TC1.5	round-trip	45.5 °C → °F → °C	≈ 45.5 °C	Pass

Reflection

This task tests temperature conversions between Celsius, Fahrenheit, and Kelvin. The main goal was to verify formula accuracy and ensure round-trip consistency (C → F → C). Unit tests confirm correct outputs and floating-point precision within a small tolerance.

Lab Task 2

Source Code
BankAccount.java:

```
package Banking;

public class BankAccount {
    private double balance;

    public void deposit(double amt) {
        if (amt < 0) throw new IllegalArgumentException("Negative deposit not allowed");
        balance += amt;
    }

    public void withdraw(double amt) {
        if (amt > balance) throw new IllegalStateException("Insufficient balance");
```

```java
        balance -= amt;
    }

    public double getBalance() {
        return balance;
    }
}
```

BankAccountTest.java:

```java
package Banking;

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

public class BankAccountTest {
    private final BankAccount account = new BankAccount();

    @Test
    void testDepositPositive() {
        account.deposit(100);
        assertEquals(100, account.getBalance());
    }

    @Test
    void testWithdrawValid() {
        account.deposit(200);
        account.withdraw(50);
        assertEquals(150, account.getBalance());
    }

    @Test
    void testWithdrawBeyondBalance() {
        assertThrows(IllegalStateException.class, () -> account.withdraw(500));
    }

    @Test
    void testDepositNegative() {
        assertThrows(IllegalArgumentException.class, () -> account.deposit(-100));
    }
}
```
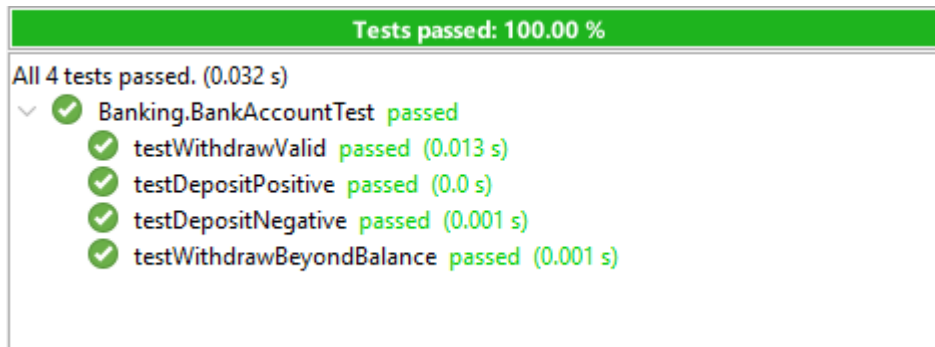
OutPut

Tests passed: 100.00 %

All 4 tests passed. (0.032 s)
  ✓ Banking.BankAccountTest passed
      ✓ testWithdrawValid passed (0.013 s)
      ✓ testDepositPositive passed (0.0 s)
      ✓ testDepositNegative passed (0.001 s)
      ✓ testWithdrawBeyondBalance passed (0.001 s)

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC2.1	deposit	100	balance = 100	Pass
TC2.2	withdraw	deposit 200 → withdraw 50	balance = 150	Pass
TC2.3	withdraw	withdraw 500 (from 0 balance)	IllegalStateException	Pass
TC2.4	deposit	deposit −100	IllegalArgumentException	Pass

Reflection

The objective was to test deposits, withdrawals, and exception handling for invalid transactions. The program ensures no negative deposits or overdrafts occur. JUnit tests validate that exceptions are correctly thrown and account balances update properly.

Lab Task 3

Source Code
StringUtil.java:

package StringUtilities;

```java
public class StringUtil {
    public boolean isPalindrome(String s) {
        if (s == null || s.isEmpty()) return false;
        String clean = s.replaceAll("[^A-Za-z]", "").toLowerCase();
        return new StringBuilder(clean).reverse().toString().equals(clean);
    }
}
```

StringUtilTest .java:

```java
package StringUtilities;

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

public class StringUtilTest {
    private final StringUtil util = new StringUtil();

    @Test
    void testSimplePalindrome() {
        assertTrue(util.isPalindrome("madam"));
    }

    @Test
    void testCaseInsensitive() {
        assertTrue(util.isPalindrome("RaceCar"));
    }

    @Test
    void testNonPalindrome() {
        assertFalse(util.isPalindrome("hello"));
    }

    @Test
    void testNullOrEmpty() {
        assertFalse(util.isPalindrome(null));
        assertFalse(util.isPalindrome(""));
    }
}
```
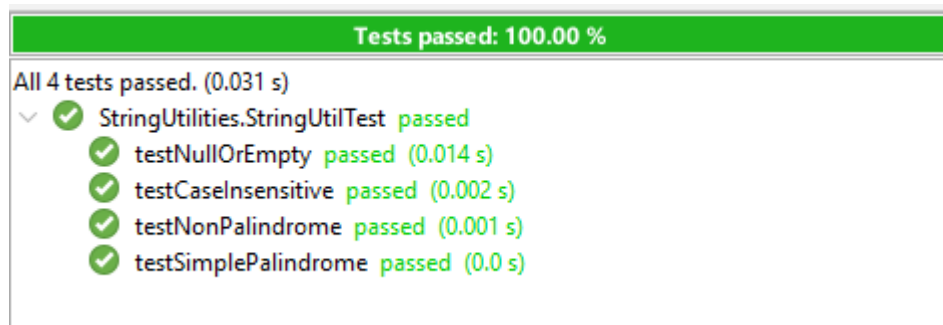
OutPut


```
Tests passed: 100.00 %
```
All 4 tests passed. (0.031 s)
- ✓ StringUtilities.StringUtilTest passed
  - ✓ testNullOrEmpty passed (0.014 s)
  - ✓ testCaseInsensitive passed (0.002 s)
  - ✓ testNonPalindrome passed (0.001 s)
  - ✓ testSimplePalindrome passed (0.0 s)

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC3.1	isPalindrome	"madam"	true	Pass
TC3.2	isPalindrome	"RaceCar"	true	Pass
TC3.3	isPalindrome	"hello"	false	Pass
TC3.4	isPalindrome	null / ""	false	Pass

Reflection

This task checks if a string reads the same forward and backward, ignoring case and punctuation. The test suite covers normal, mixed-case, and invalid inputs (null or empty). It demonstrates string cleaning and logic verification using JUnit assertions.

Lab Task 4

Source Code
TimerUtil .java:

package Timer;

public class TimerUtil {
    public int secondsBetween(int start, int end) {
        if (end < start) throw new IllegalArgumentException("End < start");

```
        return end - start;
    }
}
```

TimerUtilTest .java:

```java
package Timer;

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

public class TimerUtilTest {
    private final TimerUtil timer = new TimerUtil();

    @Test
    void testNormalCase() {
        assertEquals(15, timer.secondsBetween(10, 25));
    }

    @Test
    void testBoundaryCase() {
        assertEquals(0, timer.secondsBetween(0, 0));
    }

    @Test
    void testInvalidInput() {
        assertThrows(IllegalArgumentException.class, () -> timer.secondsBetween(20, 10));
    }
}
```
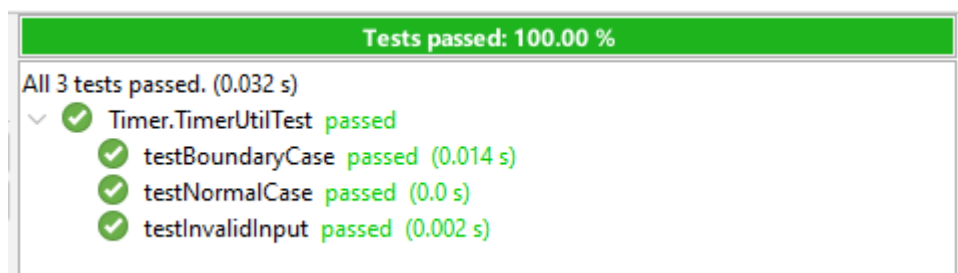
OutPut

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC4.1	secondsBetween	(10, 25)	15 s	Pass
TC4.2	secondsBetween	(0, 0)	0 s	Pass
TC4.3	secondsBetween	(20, 10)	IllegalArgumentException	Pass

Reflection

The Timer Utility computes elapsed seconds between two timestamps. Tests focus on normal cases, zero-length intervals, and invalid input (end < start). Exception testing ensures robust input validation and reliable function behavior.

Lab Task 5

Source Code
ShoppingCart .java:

```java
package Shopping;

import java.util.*;
public class ShoppingCart {
    private final List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public void removeItem(String item) {
        items.remove(item);
    }

    public int getItemCount() {
        return items.size();
    }
```

```java
    public void clear() {
        items.clear();
    }
}
```

ShoppingCartTest.java:

```java
package Shopping;

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

public class ShoppingCartTest {
    private final ShoppingCart cart = new ShoppingCart();

    @Test
    void testAddItems() {
        cart.addItem("A");
        cart.addItem("B");
        cart.addItem("C");
        assertEquals(3, cart.getItemCount());
    }

    @Test
    void testRemoveItem() {
        cart.addItem("X");
        cart.addItem("Y");
        cart.removeItem("Y");
        assertEquals(1, cart.getItemCount());
    }

    @Test
    void testClearCart() {
        cart.addItem("A");
        cart.addItem("B");
        cart.clear();
        assertEquals(0, cart.getItemCount());
    }

    @Test
    void testRemoveNonExistentItem() {
        assertDoesNotThrow(() -> cart.removeItem("Z"));
    }
```

}

OutPut
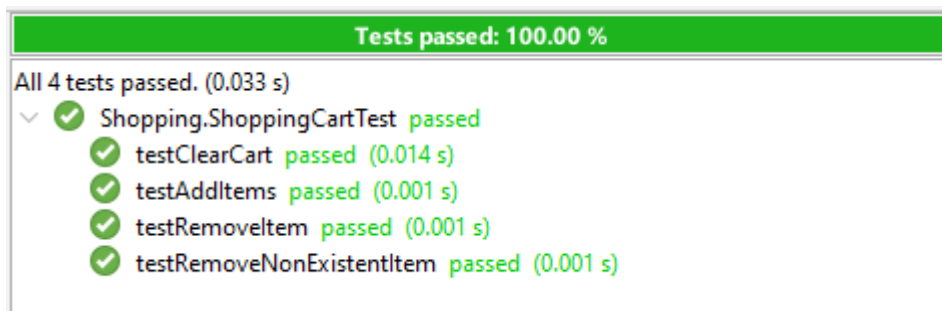
TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC5.1	addItem / getItemCount	Add 3 items	count = 3	Pass
TC5.2	removeItem	Remove 1 of 3 items	count = 2	Pass
TC5.3	clear	Clear cart	count = 0	Pass
TC5.4	removeItem	Remove non-existent item	No error thrown	Pass

Reflection

This task simulates basic cart operations—adding, removing, and clearing items. Tests confirm item counts update correctly and that removing non-existent items doesn't cause errors. It demonstrates testing for state consistency in list-based objects.

Home Work 1

Source Code
Calculator1 .java:

```java
package Calculator;

public class Calculator1 {

    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;
    }

    public int divide(int a, int b) {
        if (b == 0) throw new IllegalArgumentException("Cannot divide by zero.");
        return a / b;
    }

    public double power(int base, int exp) {
        return Math.pow(base, exp);
    }

    public int modulus(int a, int b) {
        if (b == 0) throw new IllegalArgumentException("Modulus by zero not allowed.");
        return a % b;
    }
}
```

Calculator1Test .java:

```java
package Calculator;

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

public class Calculator1Test {

    private Calculator1 calc;

    @BeforeEach
    void setUp() {
```

```java
        calc = new Calculator1();
    }

    @AfterEach
    void tearDown() {
        calc = null;
    }

    @Test
    void testBasicOperations() {
        assertEquals(10, calc.add(7, 3));
        assertEquals(4, calc.subtract(7, 3));
        assertEquals(21, calc.multiply(7, 3));
        assertEquals(2, calc.divide(10, 5));
    }

    @Test
    void testPower() {
        assertEquals(8.0, calc.power(2, 3), 0.001);
        assertEquals(1.0, calc.power(5, 0), 0.001);
    }

    @Test
    void testModulus() {
        assertEquals(1, calc.modulus(10, 3));
    }

    @Test
    void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> calc.divide(5, 0));
    }

    @Test
    void testModulusByZero() {
        assertThrows(IllegalArgumentException.class, () -> calc.modulus(9, 0));
    }
}
```
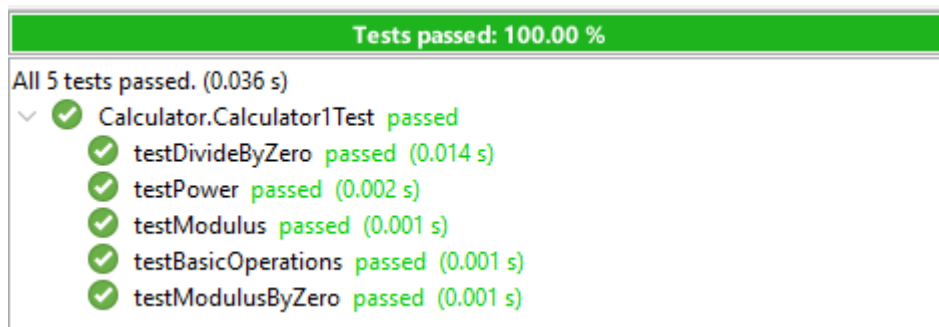
OutPut

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC6.1	add	7, 3	10	Pass
TC6.2	power	2, 3	8.0	Pass
TC6.3	modulus	10, 3	1	Pass
TC6.4	divide	5, 0	IllegalArgumentException	Pass
TC6.5	modulus	9, 0	IllegalArgumentException	Pass

Reflection

The enhanced calculator adds power and modulus operations to a basic arithmetic class. Tests verify correct results for all operations, including exceptions for division and modulus by zero. Lifecycle annotations (@BeforeEach, @AfterEach) demonstrate structured test initialization and cleanup.

Home Work 2

Source Code
BankAccount2 .java:

package Banking;

```java
public class BankAccount2 {
    private double balance;
    private boolean active = true;

    public void deposit(double amt) {
        if (amt < 0) throw new IllegalArgumentException("Negative deposit not allowed");
        balance += amt;
        checkActivity();
    }

    public void withdraw(double amt) {
        if (amt > balance) throw new IllegalStateException("Insufficient balance");
        balance -= amt;
        checkActivity();
    }

    public double getBalance() {
        return balance;
    }

    public boolean isActive() {
        return active;
    }

    private void checkActivity() {
        active = balance >= 100;
    }
}
```

BankAccount2Test .java:

```java
package Banking;

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

public class BankAccount2Test {

    @Test
    void testValidDeposit() {
        BankAccount2 acc = new BankAccount2();
        acc.deposit(500);
        assertEquals(500, acc.getBalance());
        assertTrue(acc.isActive());
```

```java
    }

    @Test
    void testOverdraft() {
        BankAccount2 acc = new BankAccount2();
        assertThrows(IllegalStateException.class, () -> acc.withdraw(50));
    }

    @Test
    void testNegativeDeposit() {
        BankAccount2 acc = new BankAccount2();
        assertThrows(IllegalArgumentException.class, () -> acc.deposit(-100));
    }

    @Test
    void testInactivityAfterWithdrawal() {
        BankAccount2 acc = new BankAccount2();
        acc.deposit(150);
        acc.withdraw(100);
        assertFalse(acc.isActive());
    }

    @Test
    void testMultipleTransactions() {
        BankAccount2 acc = new BankAccount2();
        acc.deposit(200);
        acc.withdraw(50);
        assertEquals(150, acc.getBalance());
        assertTrue(acc.isActive());
    }
}
```
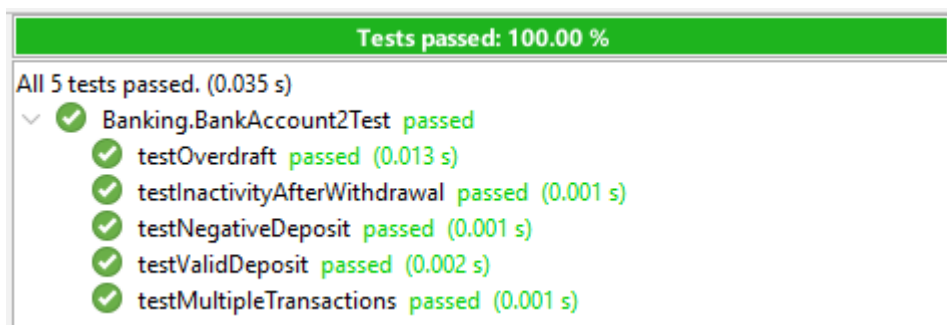
OutPut

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC7.1	deposit	+500	balance = 500, active = true	Pass
TC7.2	withdraw	withdraw 50 (from 0 balance)	IllegalStateException	Pass
TC7.3	deposit	−100	IllegalArgumentException	Pass
TC7.4	withdraw	deposit 150 → withdraw 100	active = false	Pass
TC7.5	multiple ops	deposit 200 → withdraw 50	balance = 150, active = true	Pass

Reflection

This version of BankAccount introduces an "active" status based on balance thresholds. Tests confirm correct balance updates, inactivity when funds fall below 100, and proper exception handling for invalid deposits or overdrafts. It connects business logic with rigorous unit testing.

Home Work 3

Source Code
StringAnalyzer .java:

```
package StringUtilities;

import java.util.Arrays;
```

```java
public class StringAnalyzer {

    public boolean isPalindrome(String s) {
        if (s == null || s.isEmpty()) return false;
        String clean = s.replaceAll("[^A-Za-z]", "").toLowerCase();
        return new StringBuilder(clean).reverse().toString().equals(clean);
    }

    public int countVowels(String s) {
        if (s == null) return 0;
        int count = 0;
        for (char c : s.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) count++;
        }
        return count;
    }

    public boolean isAnagram(String s1, String s2) {
        if (s1 == null || s2 == null) return false;
        char[] a = s1.replaceAll("[^A-Za-z]", "").toLowerCase().toCharArray();
        char[] b = s2.replaceAll("[^A-Za-z]", "").toLowerCase().toCharArray();
        Arrays.sort(a);
        Arrays.sort(b);
        return Arrays.equals(a, b);
    }
}
```

StringAnalyzerTest .java:

```java
package StringUtilities;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class StringAnalyzerTest {

    private final StringAnalyzer analyzer = new StringAnalyzer();

    @Test
    @DisplayName("Check palindrome detection")
    void testPalindrome() {
        assertTrue(analyzer.isPalindrome("Level"));
```
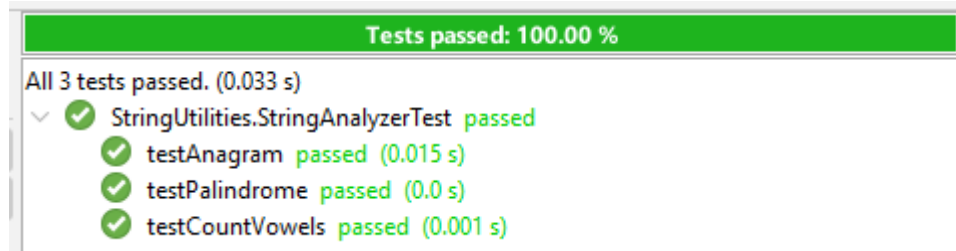
```java
      assertFalse(analyzer.isPalindrome("Java"));
  }

  @Test
  @DisplayName("Count vowels in a string")
  void testCountVowels() {
     assertEquals(2, analyzer.countVowels("Hello"));
     assertEquals(0, analyzer.countVowels(""));
     assertEquals(0, analyzer.countVowels(null));
  }

  @Test
  @DisplayName("Check if two words are anagrams")
  void testAnagram() {
     assertTrue(analyzer.isAnagram("listen", "silent"));
     assertFalse(analyzer.isAnagram("test", "text"));
     assertFalse(analyzer.isAnagram(null, "text"));
  }
}
```

OutPut

	Tests passed: 100.00 %

All 3 tests passed. (0.033 s)
- ✓ StringUtilities.StringAnalyzerTest passed
  - ✓ testAnagram passed (0.015 s)
  - ✓ testPalindrome passed (0.0 s)
  - ✓ testCountVowels passed (0.001 s)

TestCase Table

Test Case ID	Method	Input	Expected Output	Result
TC8.1	isPalindrome	"Level"	true	Pass
TC8.2	isPalindrome	"Java"	false	Pass
TC8.3	countVowels	"Hello"	3	Pass
TC8.4	countVowels	null / ""	0	Pass
TC8.5	isAnagram	"listen", "silent"	true	Pass

TC8.6	isAnagram	"test", "text"	false	Pass
TC8.7	isAnagram	null, "text"	false	Pass

Reflection

This utility performs three operations: palindrome checking, vowel counting, and anagram detection. The tests include case-insensitive checks, null handling, and various input combinations. It emphasizes robust text processing and comprehensive unit test coverage using descriptive test names.

Conclusion

This lab provided hands-on experience with **JUnit unit testing** by testing a variety of Java classes covering mathematical operations, banking logic, string processing, time calculation, and collection handling. Through writing and executing test cases, we learned how to validate correct functionality, handle edge cases, and verify exception behavior. Overall, the lab strengthened our understanding of test-driven thinking, improved code reliability, and demonstrated the importance of systematic unit testing in software quality assurance.