
# Report for Lab Tasks- JUnit Testing :

Course Code: CSE430

Course Title: Software Testing & Quality Assurance

Section: 01

## Submitted To:

Dr. Shamim H Ripon
Professor
Department of Computer Science & Engineering

## Submitted By:

Umme Mukaddisa

ID: 2022-3-60-317

## Lab Task 01- (Temperature Converter):

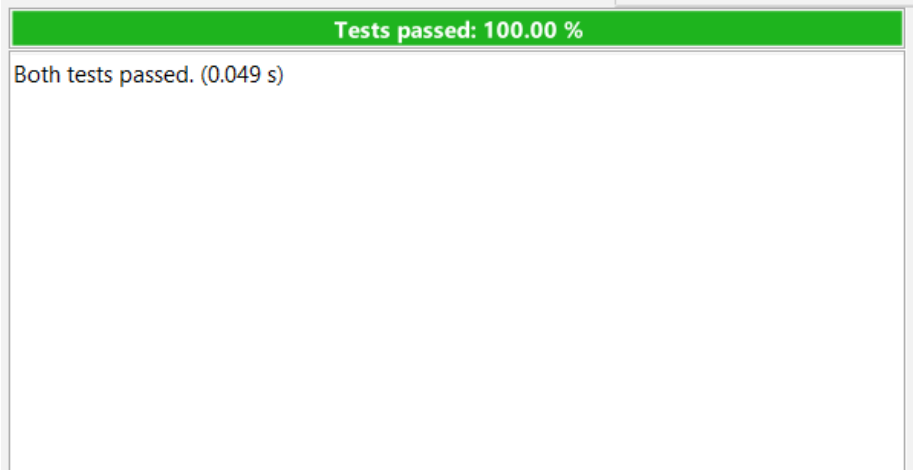
### Source Code:

**TemperatureConverter.java:**

```
package tempconverter;
public class TemperatureConverter {
    public double celsiusToFahrenheit(double c) {
        return (c * 9.0/5.0) + 32.0;
    }

    public double fahrenheitToCelsius(double f) {
        return (f - 32.0) * 5.0/9.0;
    }

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

**TemperatureConverterTest.java:**

```
package tempconverter;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TemperatureConverterTest {
    private final TemperatureConverter tc = new
TemperatureConverter();
    @Test
    void knownValues() {
        assertEquals(32.0, tc.celsiusToFahrenheit(0), 0.0001);
        assertEquals(212.0, tc.celsiusToFahrenheit(100), 0.0001);
        assertEquals(273.15, tc.celsiusToKelvin(0), 0.0001);
    }
    @Test
    void roundTrip() {
        double c = 37.5;
        double f = tc.celsiusToFahrenheit(c);
        double back = tc.fahrenheitToCelsius(f);
        assertEquals(c, back, 0.01); // tolerance for floating point
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | celsiusToFahrenheit | 0°C | 32.0°F | Verify freezing point conversion |
| 2 | celsiusToFahrenheit | 100°C | 212.0°F | Verify boiling point conversion |
| 3 | fahrenheitToCelsius | 32°F | 0°C | Reverse conversion accuracy |
| 4 | celsiusToKelvin | 0°C | 273.15 K | Validate Celsius to Kelvin conversion |
| 5 | roundTrip | 37.5°C → F → C | ~37.5°C | Ensure bidirectional consistency |

**Reflection:**

This program converts temperature values between Celsius, Fahrenheit, and Kelvin. JUnit tests verify correct conversions and ensure accurate round-trip calculations.

## Lab Task 02- (Bank Account Operations):

## Source Code:

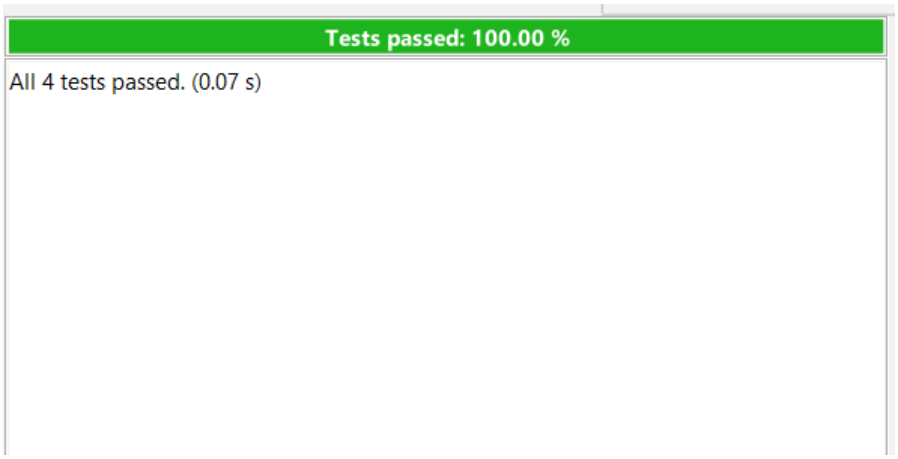
**BankAccount.java:**

```
package bankAccount;

public class BankAccount {
    private double balance;

    public BankAccount() { this.balance = 0.0; }

    public void deposit(double amt){
        if (amt < 0) throw new IllegalArgumentException("Negative
deposit");
        balance += amt;
    }

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

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

**BankAccountTest.java:**

```
package bankAccount;

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

public class BankAccountTest {
    @Test
    void depositIncreasesBalance() {
        BankAccount a = new BankAccount();
        a.deposit(200.0);
        assertEquals(200.0, a.getBalance(), 0.0001);
    }

    @Test
    void withdrawDecreasesBalance() {
        BankAccount a = new BankAccount();
```

```
        a.deposit(300.0);
        a.withdraw(100.0);
        assertEquals(200.0, a.getBalance(), 0.0001);
    }

    @Test
    void withdrawBeyondBalanceThrows() {
        BankAccount a = new BankAccount();
        a.deposit(50.0);
        assertThrows(IllegalStateException.class, () ->
a.withdraw(100.0));
    }

    @Test
    void negativeDepositThrows() {
        BankAccount a = new BankAccount();
        assertThrows(IllegalArgumentException.class, () ->
a.deposit(-20.0));
    }
}
```



**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | deposit | +200 | Balance = 200 | Normal deposit increases balance |
| 2 | withdraw | 300 deposit → withdraw 100 | Balance = 200 | Verify withdrawal works |
| 3 | withdraw | 50 deposit → withdraw 100 | Exception (Insufficient funds) | Check overdraft restriction |
| 4 | deposit | −20 | Exception (Negative deposit) | Ensure invalid deposit blocked |

**Reflection:**

Simulates a simple bank account that supports deposits and withdrawals. The test cases confirm correct balance updates and handle invalid operations like overdrafts or negative deposits.

## Lab Task 03-(String Utility (Palindrome Checker)):

## Source Code:

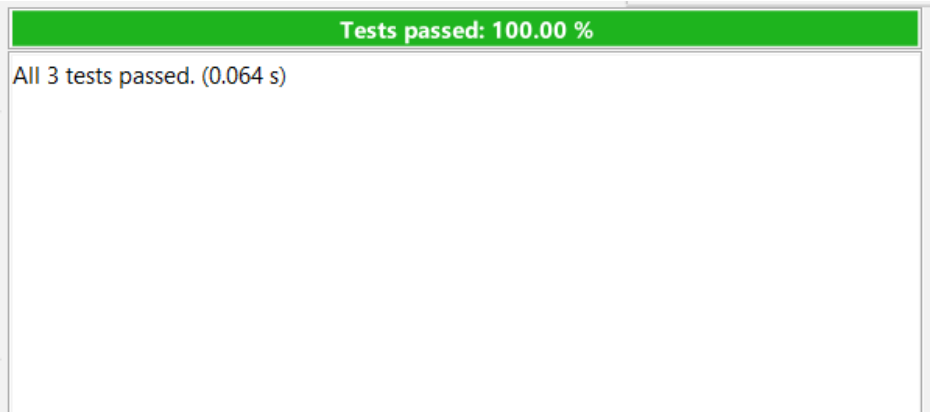
**StringUtil.java:**

```
package StringUtil;

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

```
    }
}
```

**StringUtilTest.java:**

```
package StringUtil;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;



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

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

    @Test
    void simplePalindromes() {
        assertTrue(su.isPalindrome("madam"));
        assertTrue(su.isPalindrome("RaceCar"));
    }

    @Test
    void nonPalindrome() {
        assertFalse(su.isPalindrome("hello"));
    }

    @Test
    void nullOrEmpty() {
        assertFalse(su.isPalindrome(null));
        assertFalse(su.isPalindrome(""));
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | isPalindrome | "madam" | true | Standard palindrome |
| 2 | isPalindrome | "RaceCar" | true | Case-insensitive check |
| 3 | isPalindrome | "hello" | false | Non-palindrome case |
| 4 | isPalindrome | null | false | Null input handling |
| 5 | isPalindrome | "" | false | Empty string handling |

**Reflection:**

Checks whether a given string is a palindrome while ignoring case and spaces. JUnit tests confirm correct detection and handle null or empty inputs safely.

## Lab Task 04- (Simple Timer Utility):

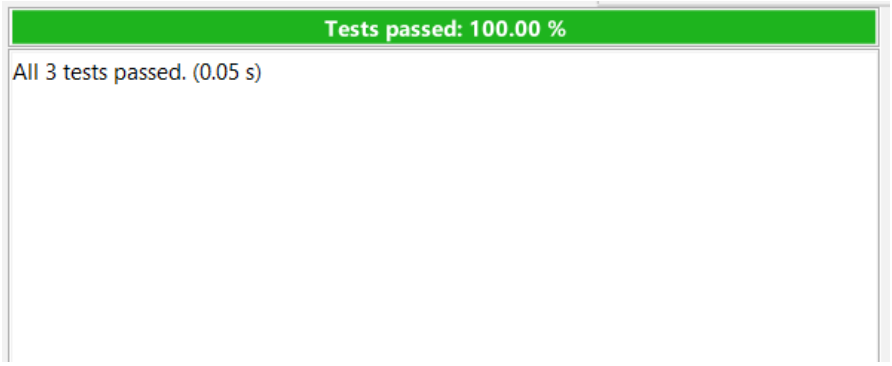
## Source Code:

**TimerUtil.java:**

```
package TimerUtil;

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

**TimerUtilTest.java:**

```
package TimerUtil;

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

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

    @Test
    void normalCase() {
        assertEquals(15, t.secondsBetween(10, 25));
    }

    @Test
    void boundaryCase() {
        assertEquals(0, t.secondsBetween(0, 0));
    }

    @Test
    void invalidInput() {
        assertThrows(IllegalArgumentException.class, () ->
t.secondsBetween(10, 5));
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | secondsBetween | (10,25) | 15 | Normal case difference |
| 2 | secondsBetween | (0,0) | 0 | Boundary (equal inputs) |
| 3 | secondsBetween | (10,5) | Exception | Invalid input (end < start) |

**Reflection:**

Computes the time difference between two second values. Tests validate normal, boundary, and invalid (end < start) cases.

## Lab Task 05- (Shopping Cart (Mini Case Study) ):

## Source Code:

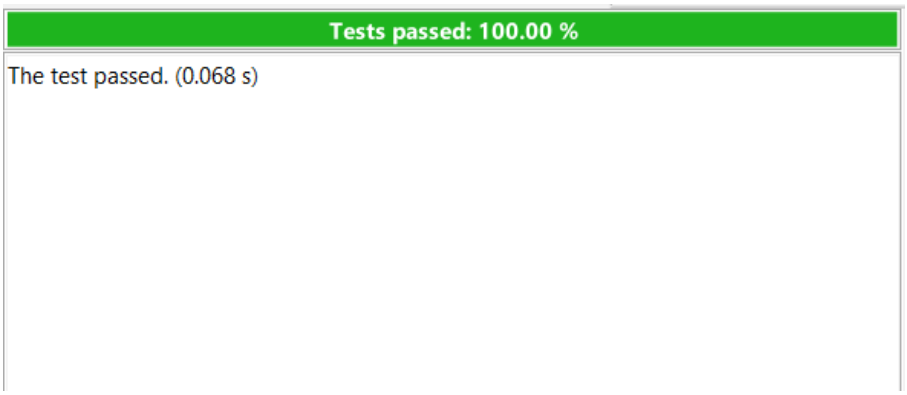
**ShoppingCart.java:**

```
package ShoppingCart;
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(); }

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

**ShoppingCartTest.java:**

```
package ShoppingCart;

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

public class ShoppingCartTest {
    @Test
    void addRemoveClear() {
        ShoppingCart c = new ShoppingCart();
        c.addItem("apple");
        c.addItem("banana");
        c.addItem("orange");
        assertEquals(3, c.getItemCount());

        c.removeItem("banana");
        assertEquals(2, c.getItemCount());

        c.removeItem("not-exist"); // should not throw
```

```
        assertEquals(2, c.getItemCount());

        c.clear();
        assertEquals(0, c.getItemCount());
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | addItem | “apple”, “banana”, “orange” | Count = 3 | Add multiple items |
| 2 | removeItem | Remove “banana” | Count = 2 | Remove existing item |
| 3 | removeItem | Remove “not-exist” | Count unchanged | Handle absent item gracefully |
| 4 | clear | — | Count = 0 | Verify clear empties cart |

**Reflection:**

Implements a simple shopping cart where users can add, remove, and clear items.Test cases confirm proper item counting and error-free handling of invalid removals.

## Homework 1- (Enhanced Calculator) :

### Source Code:

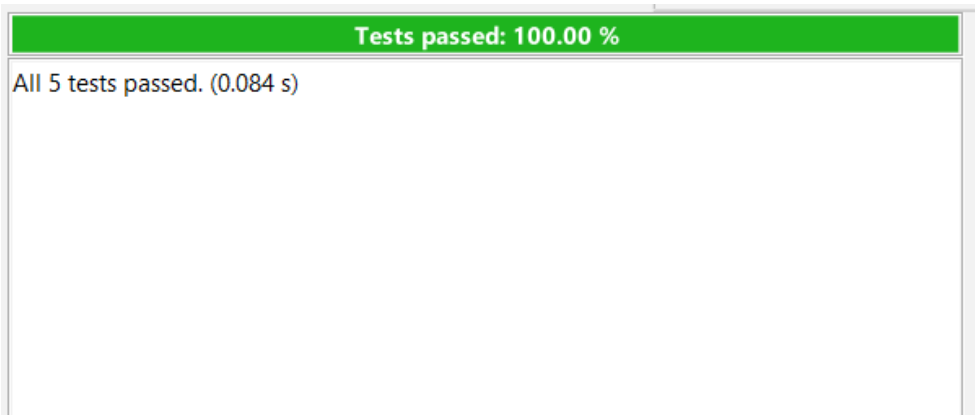
**Calculator.java:**

```
package Calculator;

public class Calculator {
    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 int power(int base, int exp) {
        if (exp < 0) throw new IllegalArgumentException("Negative
exponent not supported.");
        int res = 1;
        for (int i = 0; i < exp; i++) res *= base;
        return res;
    }

    public int modulus(int a, int b) {
        if (b == 0) throw new IllegalArgumentException("Modulus by
zero.");
        return a % b;
    }
}
```

**CalculatorTest.java:**

```
package Calculator;

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

public class CalculatorTest {
    private Calculator calc;

    @BeforeEach
    void setUp() {
        calc = new Calculator();
```

```
    }

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

    @Test
    void basicOps() {
        assertEquals(5, calc.add(2,3));
        assertEquals(4, calc.subtract(10,6));
        assertEquals(20, calc.multiply(4,5));
        assertEquals(2, calc.divide(10,5));
    }

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

    @Test
    void powerNormal() {
        assertEquals(8, calc.power(2,3));
        assertEquals(1, calc.power(5,0));
    }

    @Test
    void powerNegativeExpThrows() {
        assertThrows(IllegalArgumentException.class, () ->
calc.power(2,-1));
    }

    @Test
    void modulusNormalAndException() {
        assertEquals(1, calc.modulus(10,3));
        assertThrows(IllegalArgumentException.class, () ->
calc.modulus(10,0));
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | add | (2,3) | 5 | Normal addition |
| 2 | subtract | (10,6) | 4 | Subtraction test |
| 3 | multiply | (4,5) | 20 | Multiplication check |
| 4 | divide | (10,5) | 2 | Normal division |
| 5 | divide | (5,0) | Exception | Divide by zero handling |
| 6 | power | (2,3) | 8 | Normal exponentiation |
| 7 | power | (5,0) | 1 | Power of zero |
| 8 | power | (2,-1) | Exception | Negative exponent handling |
| 9 | modulus | (10,3) | 1 | Modulo operation |
| 10 | modulus | (10,0) | Exception | Mod by zero prevention |

**Reflection:**

Performs basic arithmetic operations with added power and modulus functions. JUnit tests ensure correct results and verify exceptions for divide-by-zero or negative exponent cases.

## Homework 2- (Account Validation):

### Source Code:

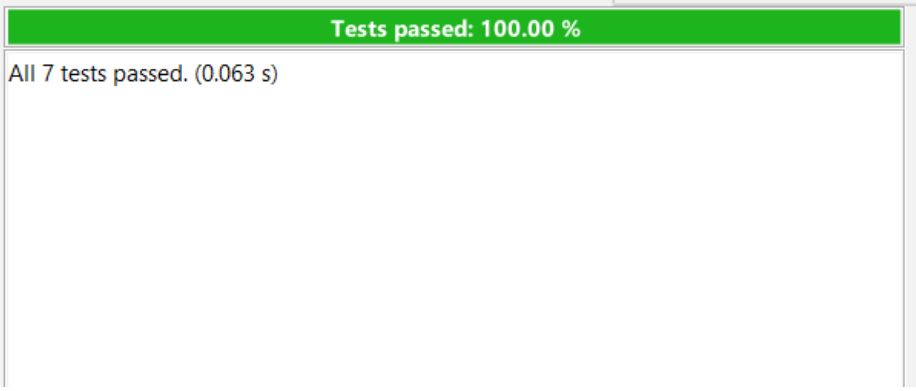
**BankAcc.java:**

```
package BankAcc;

public class BankAcc {
    private double balance;
    private boolean active;

    public BankAcc() {
        this.balance = 0.0;
        this.active = true;
    }

    public void deposit(double amt) {
        if (amt < 0) throw new IllegalArgumentException("Negative
deposit");
        balance += amt;
        if (balance >= 100.0) active = true;
    }

    public void withdraw(double amt) {
        if (amt < 0) throw new IllegalArgumentException("Negative
withdrawal");
        if (amt > balance) throw new
IllegalStateException("Insufficient funds");
        balance -= amt;
        if (balance < 100.0) active = false;
    }

    public double getBalance() { return balance; }

    public boolean isActive() { return active; }
}
```

**BankAccTest.java:**

```
package BankAcc;

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

public class BankAccTest {

    @Test
    void testValidDeposit() {
        BankAcc acc = new BankAcc();
        acc.deposit(200.0);
        assertEquals(200.0, acc.getBalance(), 0.0001);
        assertTrue(acc.isActive(), "Account should remain active
after deposit ≥ 100");
    }

    @Test
    void testDepositBelowThresholdKeepsActive() {
        BankAcc acc = new BankAcc();
        acc.deposit(50.0);
        assertEquals(50.0, acc.getBalance(), 0.0001);
        // Initially active, remains true until withdrawal rule
changes it
        assertTrue(acc.isActive());
    }

    @Test
    void testWithdrawWithinBalance() {
        BankAcc acc = new BankAcc();
        acc.deposit(300.0);
        acc.withdraw(150.0);
        assertEquals(150.0, acc.getBalance(), 0.0001);
        assertTrue(acc.isActive(), "Balance still ≥100, should remain
active");
    }

    @Test
    void testWithdrawMakesInactive() {
        BankAcc acc = new BankAcc();
        acc.deposit(300.0);
        acc.withdraw(250.0); // Leaves 50.0
        assertEquals(50.0, acc.getBalance(), 0.0001);
```

```
        assertFalse(acc.isActive(), "Balance <100, should become
inactive");
    }

    @Test
    void testWithdrawBeyondBalanceThrows() {
        BankAcc acc = new BankAcc();
        acc.deposit(50.0);
        assertThrows(IllegalStateException.class, () ->
acc.withdraw(100.0),
                "Should not allow withdrawal exceeding balance");
    }

    @Test
    void testNegativeDepositThrows() {
        BankAcc acc = new BankAcc();
        assertThrows(IllegalArgumentException.class, () ->
acc.deposit(-10.0),
                "Negative deposit should throw exception");
    }

    @Test
    void testNegativeWithdrawThrows() {
        BankAcc acc = new BankAcc();
        assertThrows(IllegalArgumentException.class, () ->
acc.withdraw(-5.0),
                "Negative withdrawal should throw exception");
    }
}
```



**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | deposit | +200 | Balance=200, Active=true | Normal deposit activates account |
| 2 | withdraw | 300→withdraw250 | Balance=50, Active=false | Account becomes inactive below threshold |
| 3 | withdraw | 50→withdraw100 | Exception | Overdraft protection |
| 4 | deposit | −10 | Exception | Invalid deposit |
| 5 | withdraw | −5 | Exception | Invalid withdrawal |

**Reflection:**

Enhances the bank account with an "active" status that depends on the balance. Tests validate activation/deactivation rules and handle negative or invalid transactions safely.

# Homework 3- (String Utility Testing):

## Source Code:

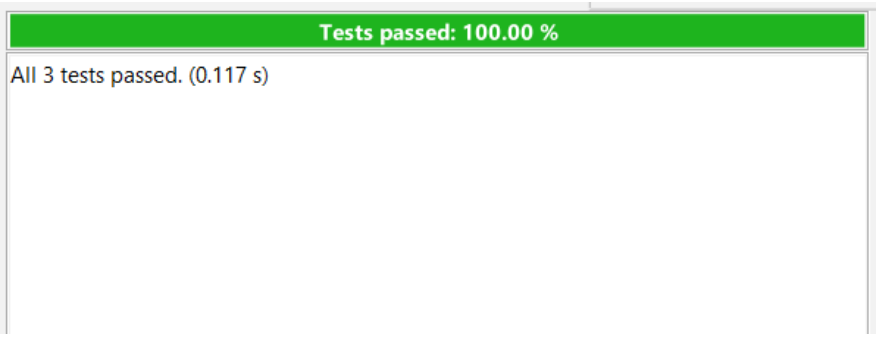
**StringUtilityAnalyzer.java:**

```
package StringUtilityAnalyzer;

import java.util.Arrays;

public class StringUtilityAnalyzer {

    public boolean isPalindrome(String s) {
        if (s == null || s.isEmpty()) return false;

        String clean = s.replaceAll("[^A-Za-z]", "").toLowerCase();
```

```
        if (clean.isEmpty()) return false;

        String reversed = new
StringBuilder(clean).reverse().toString();
        return clean.equals(reversed);
    }

    public int countVowels(String s) {
        if (s == null || s.isEmpty()) return 0;

        int count = 0;
        for (char ch : s.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(ch) >= 0) {
                count++;
            }
        }
        return count;
    }


    public boolean isAnagram(String s1, String s2) {
        if (s1 == null || s2 == null || s1.isEmpty() || s2.isEmpty())
return false;

        String a = s1.replaceAll("[^A-Za-z]", "").toLowerCase();
        String b = s2.replaceAll("[^A-Za-z]", "").toLowerCase();
        if (a.isEmpty() || b.isEmpty()) return false;

        char[] ca = a.toCharArray();
        char[] cb = b.toCharArray();
        Arrays.sort(ca);
        Arrays.sort(cb);

        return Arrays.equals(ca, cb);
    }
}
```

**StringUtilityAnalyzerTest.java:**

```
package StringUtilityAnalyzer;

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

@DisplayName("StringUtilityAnalyzer Tests")
public class StringUtilityAnalyzerTest {

    private final StringUtilityAnalyzer util = new
StringUtilityAnalyzer();

    @Test @DisplayName("Palindrome tests")
    void testPalindrome() {
        assertTrue(util.isPalindrome("madam"));
        assertTrue(util.isPalindrome("RaceCar"));
        assertFalse(util.isPalindrome("hello"));
        assertFalse(util.isPalindrome(null));
    }

    @Test @DisplayName("Vowel count tests")
    void testVowels() {
        assertEquals(2, util.countVowels("hello"));
        assertEquals(5, util.countVowels("Education"));
        assertEquals(0, util.countVowels("rhythm"));
        assertEquals(0, util.countVowels(""));
    }

    @Test @DisplayName("Anagram tests")
    void testAnagram() {
        assertTrue(util.isAnagram("Listen", "Silent"));
        assertTrue(util.isAnagram("A decimal point", "I'm a dot in
place"));
        assertFalse(util.isAnagram("hello", "bello"));
        assertFalse(util.isAnagram(null, "abc"));
    }
}
```




**Test Case Table:**

| # | Method | Input | Expected Output | Purpose |
|---|---|---|---|---|
| 1 | isPalindrome | "madam" | true | Regular palindrome |
| 2 | isPalindrome | "RaceCar" | true | Case-insensitive check |
| 3 | isPalindrome | "hello" | false | Non-palindrome |
| 4 | isPalindrome | null | false | Null check |
| 5 | countVowels | "hello" | 2 | Normal vowel count |
| 6 | countVowels | "Education" | 5 | Case-insensitive vowel count |
| 7 | countVowels | "rhythm" | 0 | No vowels |
| 8 | isAnagram | ("Listen","Silent") | true | Valid anagram |
| 9 | isAnagram | ("A decimal point","I'm a dot in place") | true | Anagram with punctuation ignored |
| 10 | isAnagram | ("hello","bello") | false | Non-anagram |
| 11 | isAnagram | (null,"abc") | false | Null input |

**Reflection:**

Provides string analysis methods: palindrome check, vowel counting, and anagram detection. Test cases confirm accuracy across normal and edge inputs, ignoring spaces, punctuation, and case.