**EXP NO: 11**

**Demonstrate the working of Junit to reverse a word and using assert statement for Proof of the value.**

package proggrammingknowledge;

public class Stringreverse {

public String reverse(String input) {

if (input == null) {

return input;

}

String output = "";

for (int i = input.length() - 1; i >= 0; i--) {

output = output + input.charAt(i);

}

return output;

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class Stringreversee {

@Test

void testStringreversee() {

Stringreverse example = new Stringreverse();

assertEquals("tac", example.reverse("cat"));

assertNull(example.reverse(null));

assertEquals("", example.reverse(""));

}

}

**OUTPUT**

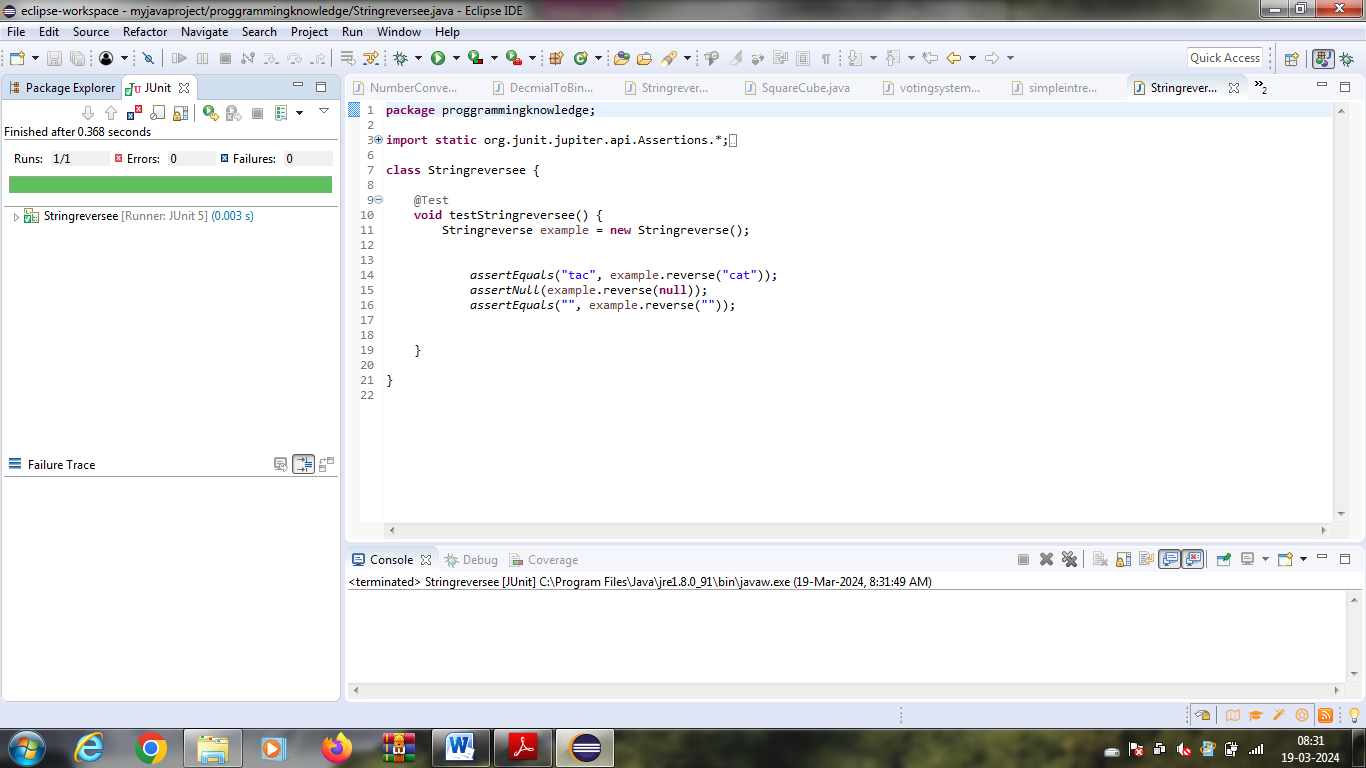
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| cat | tac |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

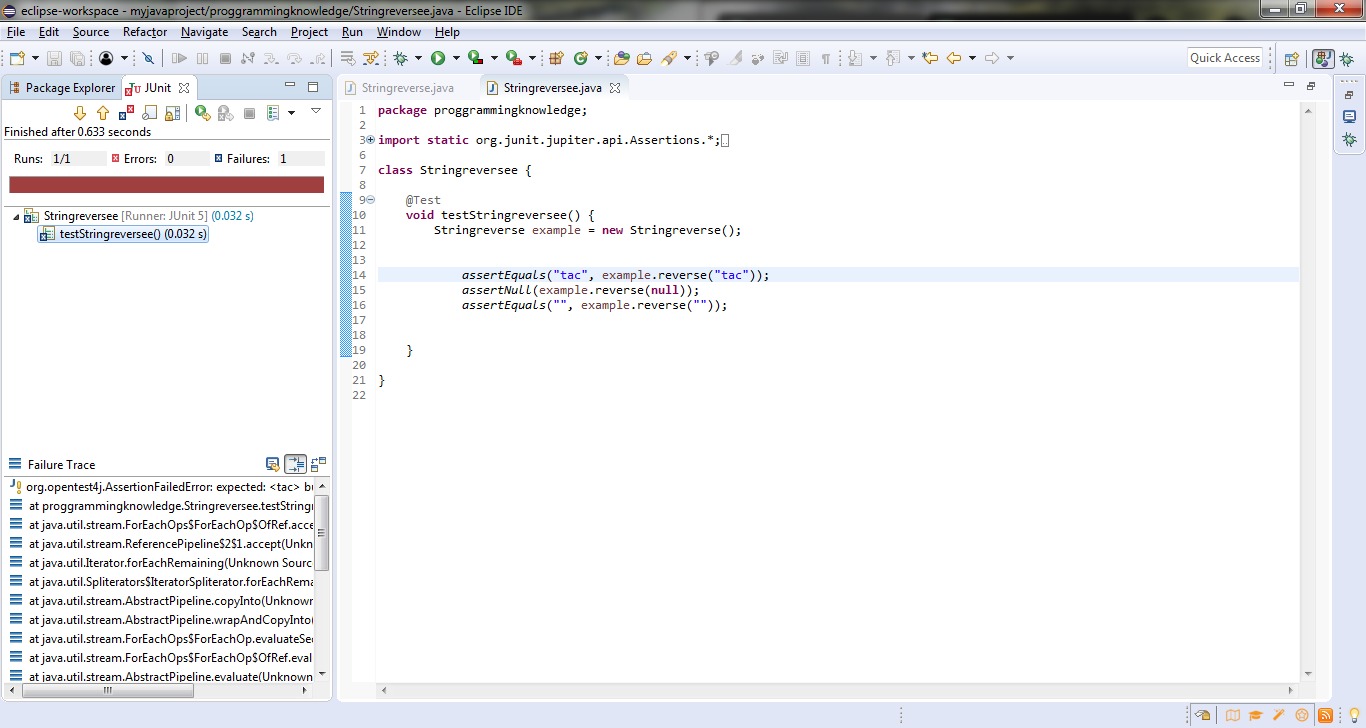
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| cat | tac | tac | Success |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| cat | Tac | Failure | FAILURE |



///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**EXP NO: 12**

**Write a white box testing code (Junit) to String comparison of word and using assert statement for Proof the value**.

package proggrammingknowledge;

public class Stringcompare {

public void compareStrings(String str1, String str2) {

if (str1.equals(str2)) {

System.out.println("Strings are equal.");

} else {

System.out.println("Strings are not equal.");

}

}

public static void main(String[] args) {

String str1 = "Hello";

String str2 = "hello";

Stringcompare comparer = new Stringcompare();

comparer.compareStrings(str1, str2);

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class Stringcomparee {

@Test

void testStringcomparee() {

String str1 = "Hello";

String str2 = "Hello";

assertEquals(str1,str2);

}

}

**OUTPUT**

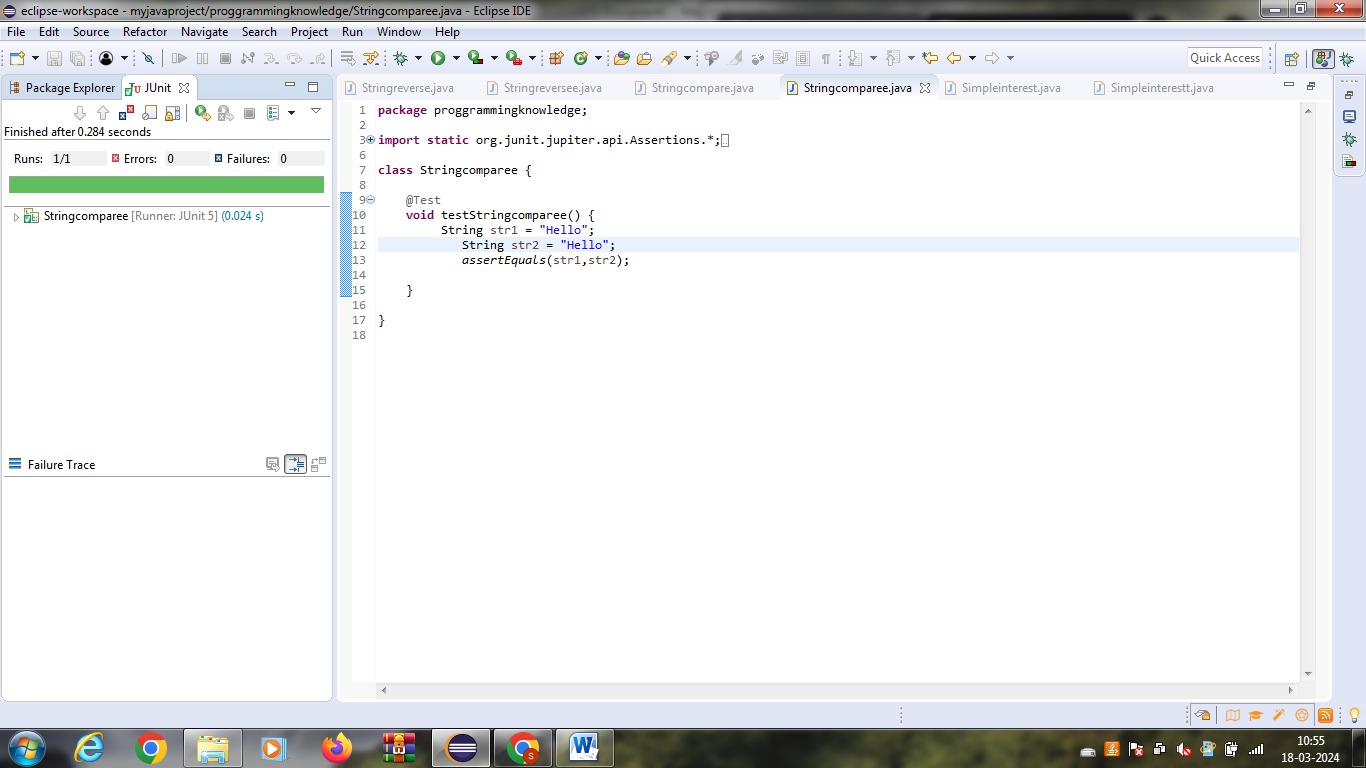
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| Hello, Hello | Hello |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

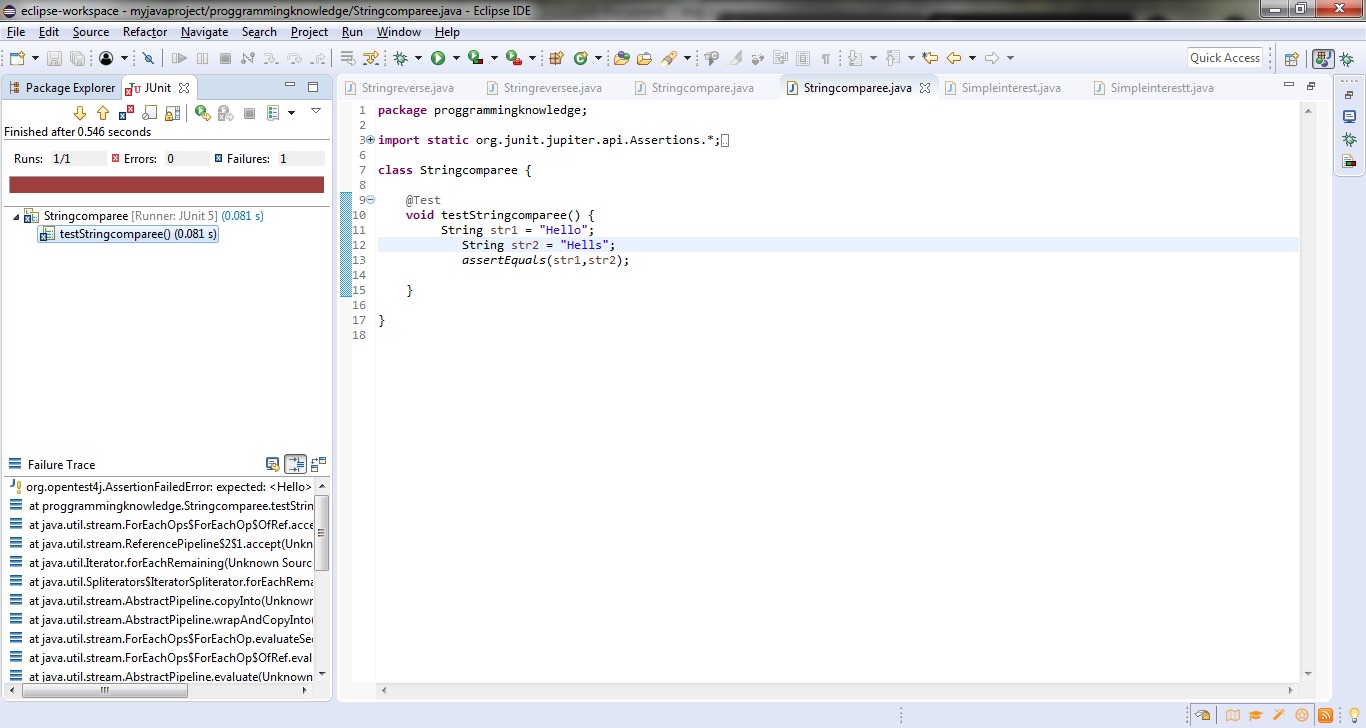
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| Hello, Hello | Hello | Hello | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| Hello, Hells | Hells | Failure | FAILURE |



//////////////////////////////////////////////////////////////////////////////////////////////////////////

**EXP NO: 13**

**Write a Junit code for voting system and uses assert statement and verify the white box testing.**

package votingsys;

public class vote {

public static String checkEligibility(int age)

{

if (age >= 18) {

return "Eligible for voting";

}

else {

return "Not eligible for voting";

}

}

}

package votingsys;

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

import org.junit.jupiter.api.Test;

class voteTest {

@Test

void testEligibleVoter() {

assertEquals("Eligible for voting", vote.checkEligibility(18));

assertEquals("Eligible for voting", vote.checkEligibility(21));

assertEquals("Eligible for voting", vote.checkEligibility(30));

}

@Test

void testNonEligibleVoter() {

assertEquals("Not eligible for voting", vote.checkEligibility(17));

assertEquals("Not eligible for voting", vote.checkEligibility(16));

assertEquals("Not eligible for voting", vote.checkEligibility(12));

}

}

**OUTPUT**

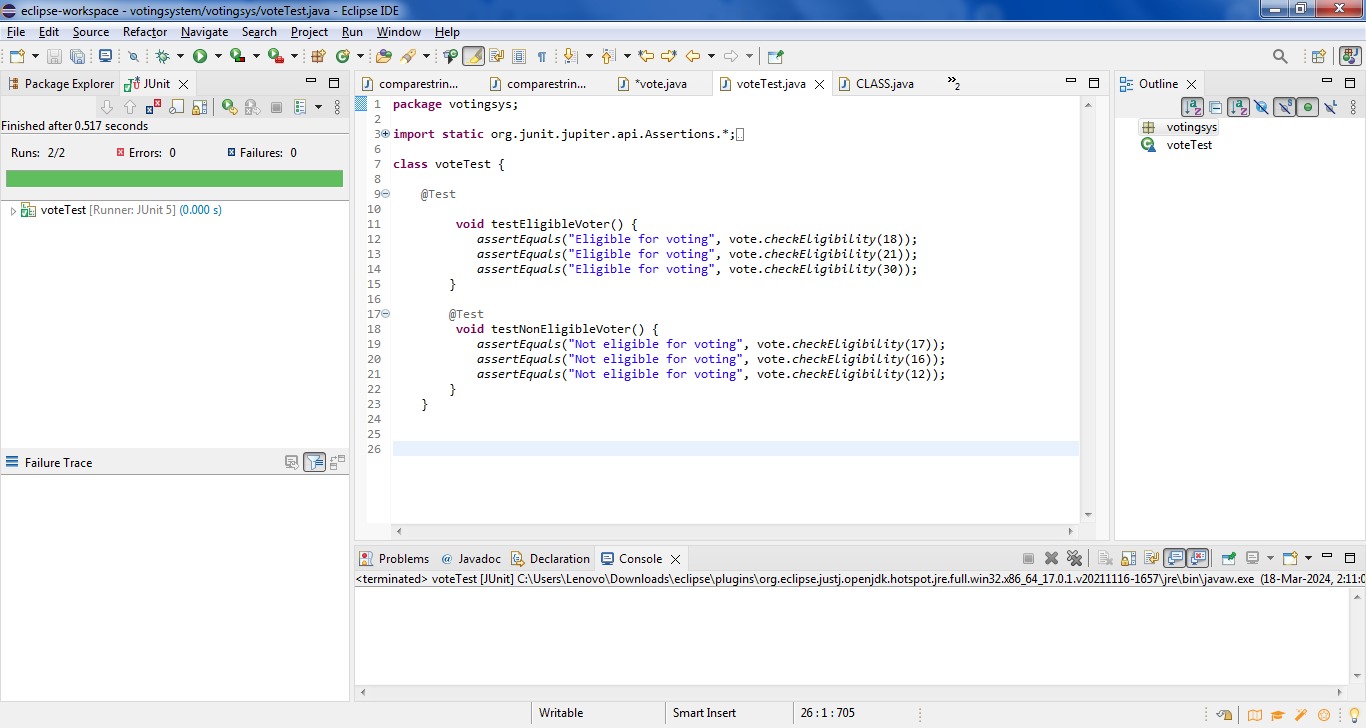
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 18 years | Eligible for vote |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

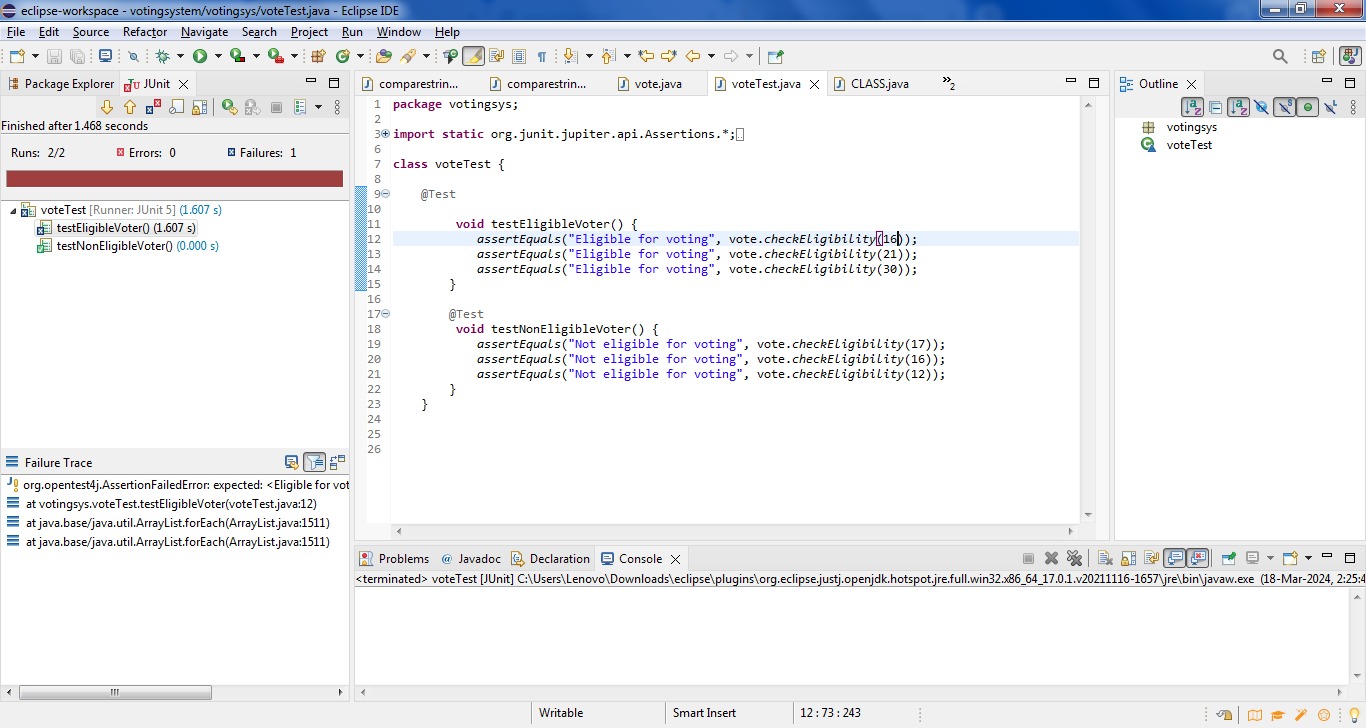
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 18 | Eligible for vote | Eligible for vote | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 16 | not eligible for vote | failure | FAILURE |

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**EX NO:14**

**Write a program using function to calculate the simple interest. Suppose the customer is a senior citizen. He is being offered 12 percent rate of interest; for all other customers, the ROI is 10 percent. The output values should verify using white box testing.**

**package** simplyintresrtt;

**public** **class** simpleintrest {

**public** **static** **double** calculateSimpleInterest(**double** principal, **double** rate, **double** time) {

**return** (principal \* rate \* time) / 100.0;

}

}

package simplyintresrtt;

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

import org.junit.jupiter.api.Test;

class simpleintrestTest {

@Test

void testCalculateSimpleInterest() {

assertEquals(200.0, simpleintrest.calculateSimpleInterest(1000.0, 5.0, 4.0), 0.01);

assertEquals(450.0, simpleintrest.calculateSimpleInterest(3000.0, 3.0, 5.0), 0.01);

assertEquals(75.0, simpleintrest.calculateSimpleInterest(1500.0, 2.0, 2.5), 0.01);

}}

**OUTPUT**

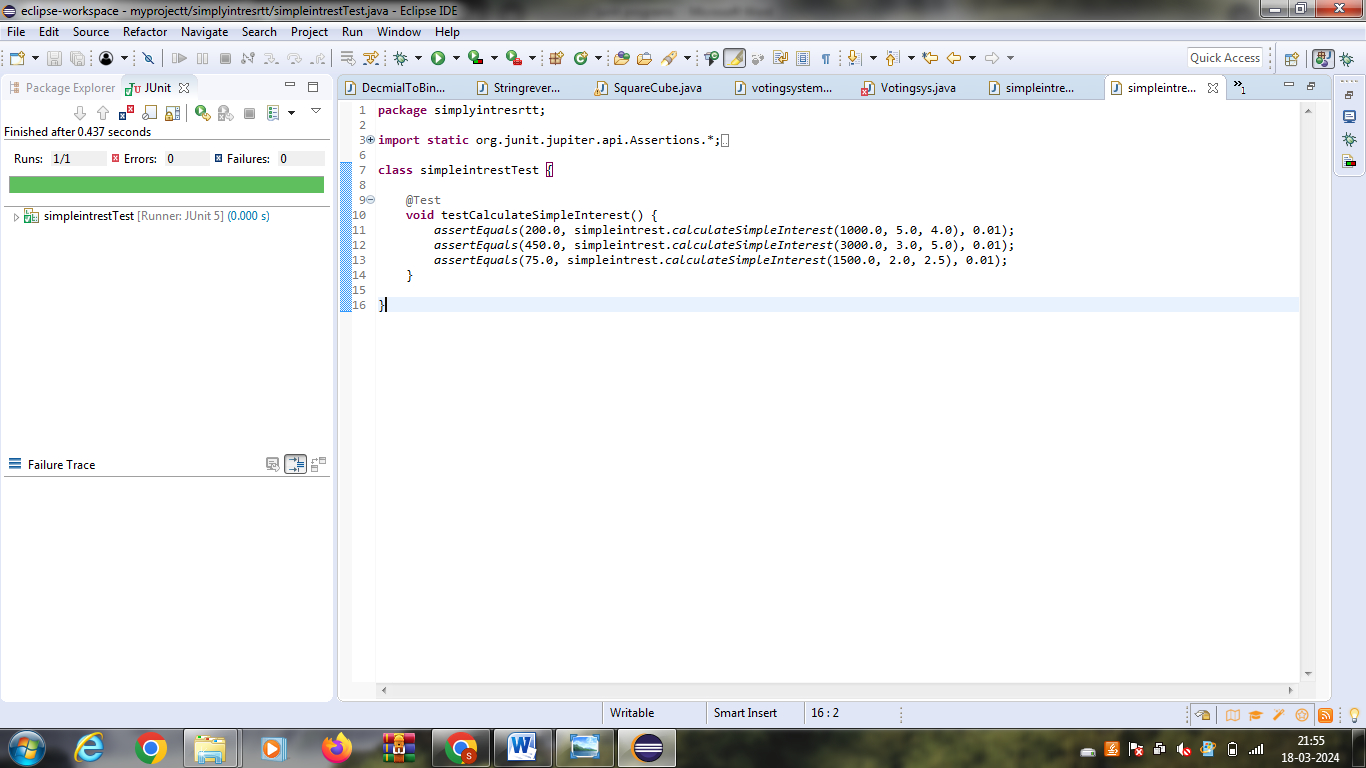
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 1500,2,2.5 | 75 |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

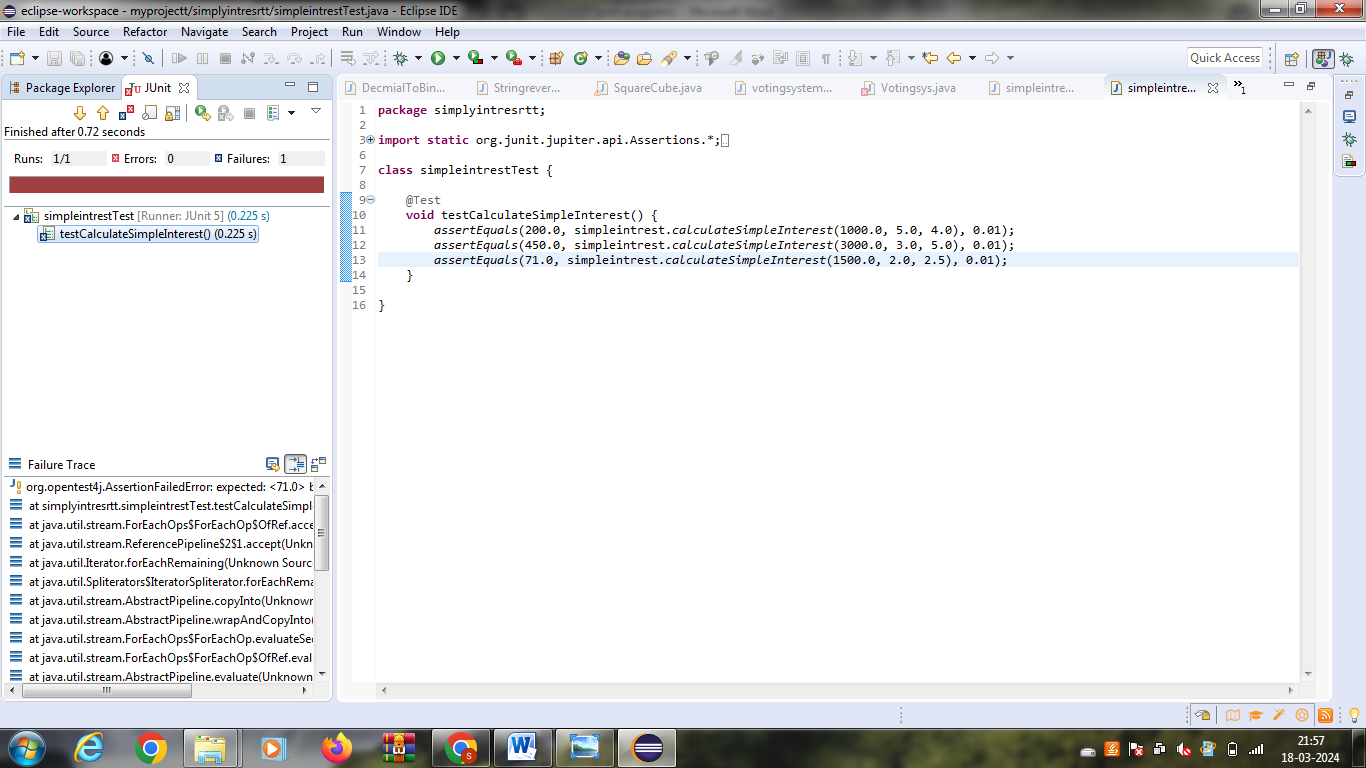
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 1500,2,2.5 | 75 | 75 | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 1500,2,2.5 | 71 | failure | FAILURE |

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**EX NO:15**

**Check whether the given number is palindrome or not and verify the output values should verify using white box testing.**

**package** proggrammingknowledge;

**public** **class** Palindrome {

**public** **static** **void** main(String[] args) {

String str = "radar";

**int** left = 0;

**int** right = str.length() - 1;

**while** (left < right && str.charAt(left) == str.charAt(right)) {

left++;

right--;

}

**if** (left >= right) {

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

} **else** {

System.***out***.println("Not Palindrome");

}

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class Palindromeornot {

@Test

void testPalindromeornot() {

String str = "radar";

assertTrue(isPalindrome(str));

}

private boolean isPalindrome(String str) {

int left = 0;

int right = str.length() - 1;

while (left < right && str.charAt(left) == str.charAt(right)) {

left++;

right--;

}

return left >= right;

}

**}**

**OUTPUT**

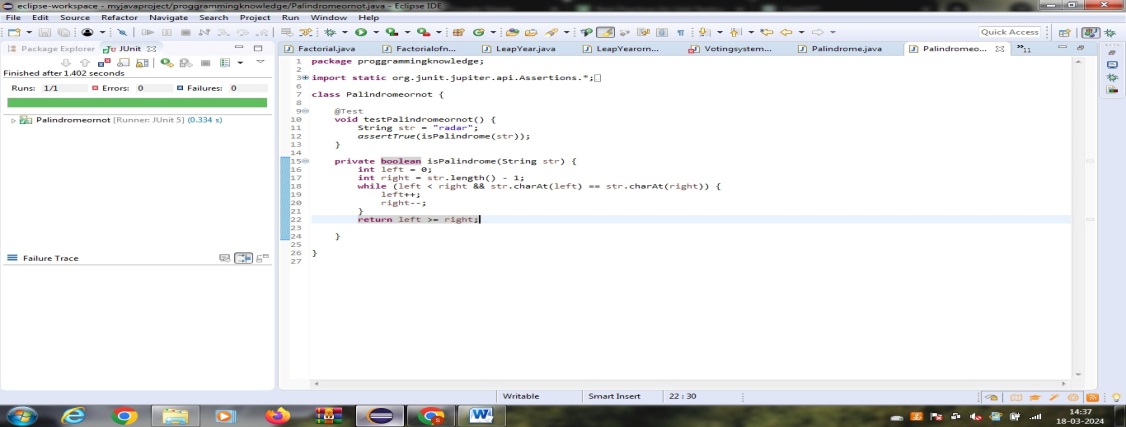
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| radar | radar |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

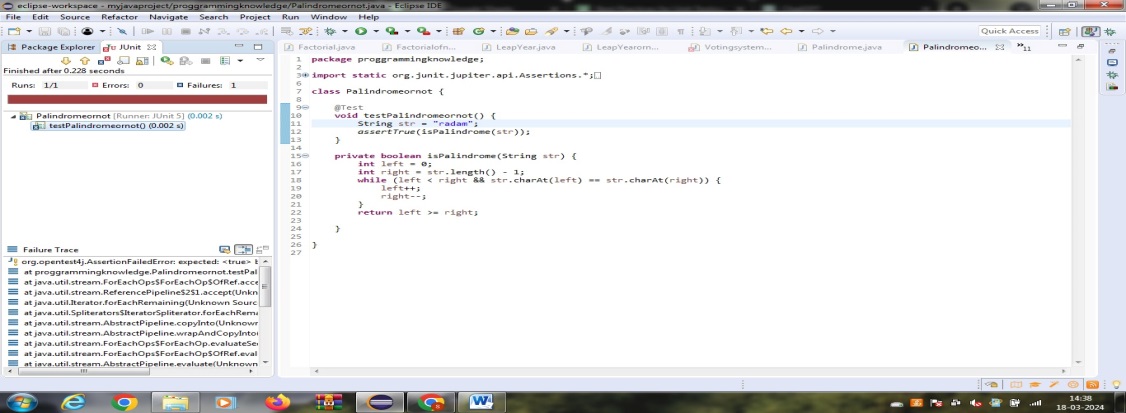
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| radar | radar | radar | SUCCESS |

****

**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| radar | radam | failure | FAILURE |

****

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**EX NO:16**

**Write a program to convert Decimal number equivalent to Binary number and octal numbers. The output values should verify using white box testing.**

**package** coding;

**public** **class** NumberConverter {

**public** **static** String decimalToBinary(**int** decimal) {

**return** Integer.*toBinaryString*(decimal);

}

**public** **static** String decimalToOctal(**int** decimal) {

**return** Integer.*toOctalString*(decimal);

}

}

package coding;

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

import org.junit.jupiter.api.Test;

class DecmialToBinaryOctal {

@Test

public void testDecimalToBinary() {

int decimal = 10;

String expectedBinary = "1010";

String actualBinary = NumberConverter.decimalToBinary(decimal);

assertEquals(expectedBinary, actualBinary);

}

@Test

public void testDecimalToOctal() {

int decimal = 10;

String expectedOctal = "12";

String actualOctal = NumberConverter.decimalToOctal(decimal);

assertEquals(expectedOctal, actualOctal);

}

}

**OUTPUT**

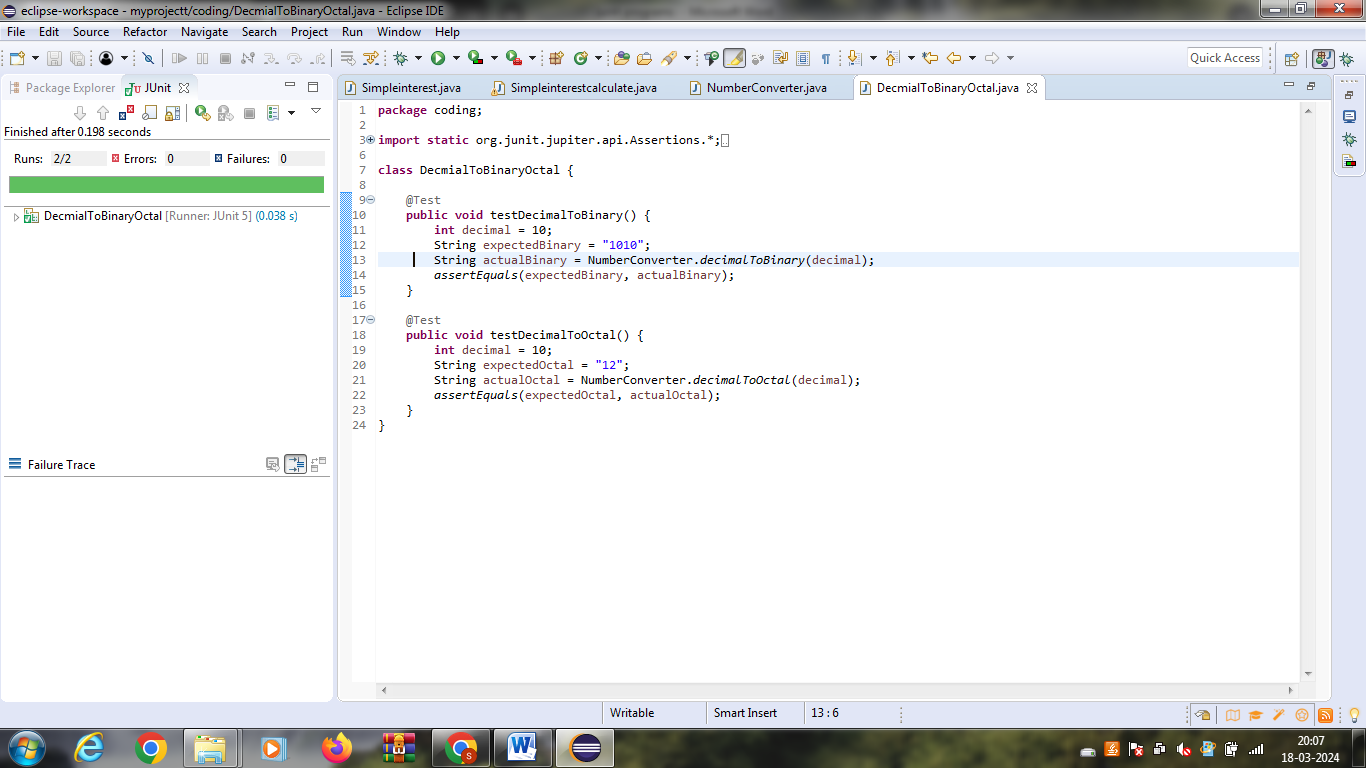
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 10 | 1010,12 |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

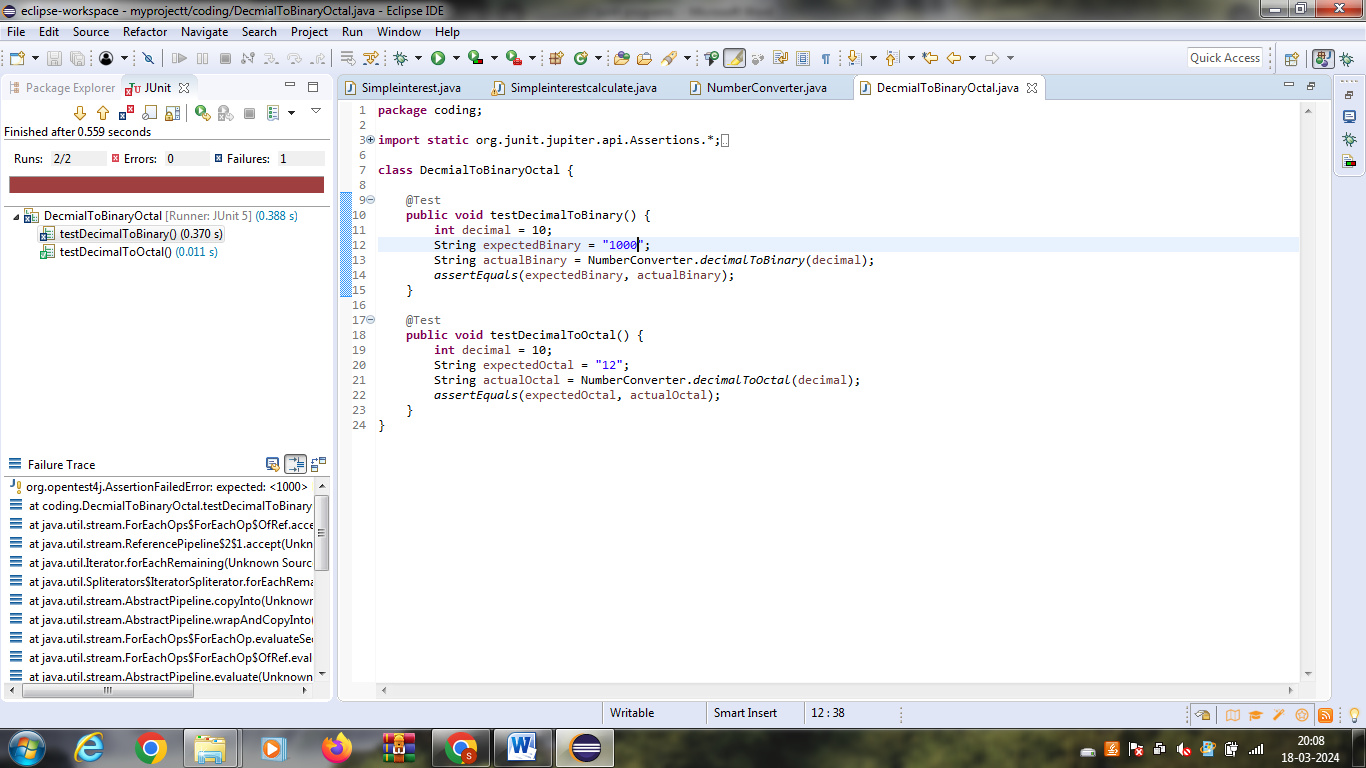
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 10 | 1010,12 | 1010,12 | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 10 | 1000,12 | Failure | FAILURE |



///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**EX NO:17**

**Write a Java Program to Convert a Given Number of Days in Terms of Years, Weeks & Days. The output values should verify using white box testing.**

**package** proggrammingknowledge;

**import** java.util.Scanner;

**public** **class** YearsWeeksDays {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of days: ");

**int** days = scanner.nextInt();

**int** years = days / 365;

**int** weeks = (days % 365) / 7;

**int** remainingDays = (days % 365) % 7;

System.***out***.println(days + " days is approximately:");

System.***out***.println(years + " years, " + weeks + " weeks, and " + remainingDays + " days.");

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class YearsWeeksDaysconvert {

@Test

void testYearsWeeksDaysconvert() {

int days = 1000;

int years = days / 365;

int weeks = (days % 365) / 7;

int remainingDays = (days % 365) % 7;

assertEquals(2, years);

assertEquals(38, weeks);

assertEquals(4, remainingDays);

}

}

**OUTPUT**

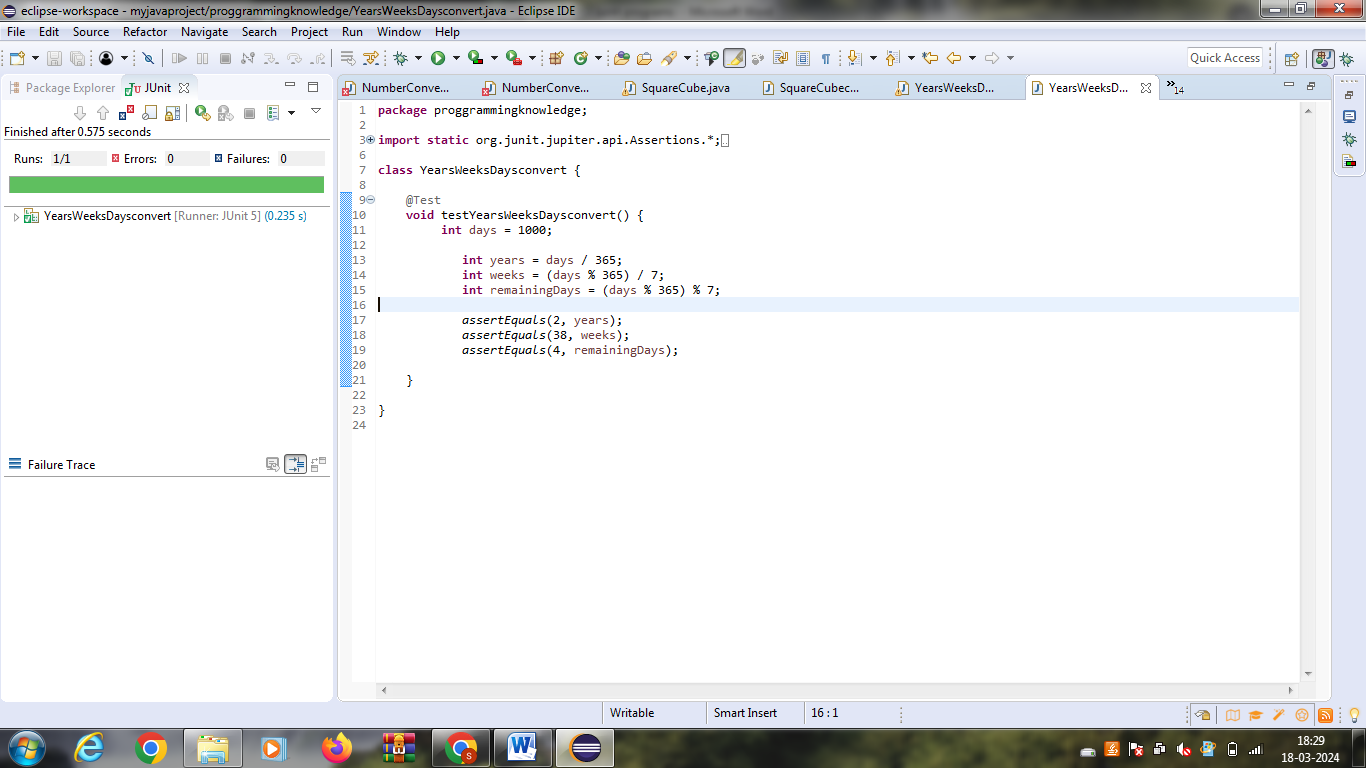
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 1000 days | 2 years,38 weeks,4 days |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

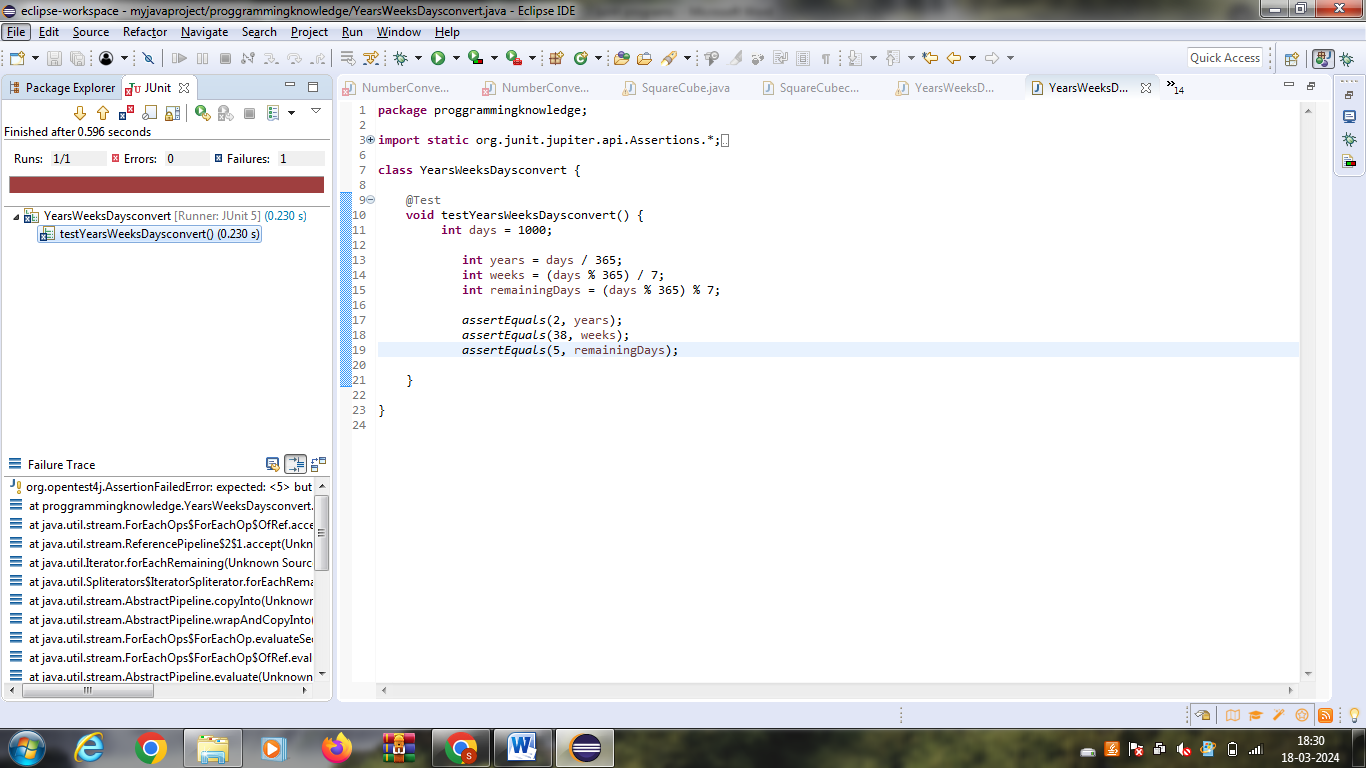
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 1000 days | 2 years,38 weeks,4 days | 2 years,38 weeks,4 days | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 1000 days | 2 years,38 weeks,5 days | failure | FAILURE |



**EX NO:18**

**Find the factorial of n. The output values should verify using white box testing.**

package proggrammingknowledge;

public class Factorial {

public static long calculateFactorial(int n) {

if (n < 0) {

throw new IllegalArgumentException("Factorial is not defined for negative numbers.");

}

if (n == 0 || n == 1) {

return 1;

}

long factorial = 1;

for (int i = 2; i <= n; i++) {

factorial \*= i;

}

return factorial;

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class Factorialofn {

@Test

void testFactorialofn() {

assertEquals(1, Factorial.calculateFactorial(0));

assertEquals(1, Factorial.calculateFactorial(1));

assertEquals(120, Factorial.calculateFactorial(5));

assertEquals(3628800, Factorial.calculateFactorial(10));

try {

Factorial.calculateFactorial(-5);

fail("Expected IllegalArgumentException");

} catch (IllegalArgumentException e) {

assertEquals("Factorial is not defined for negative numbers.", e.getMessage());

}

}

}

**OUTPUT**

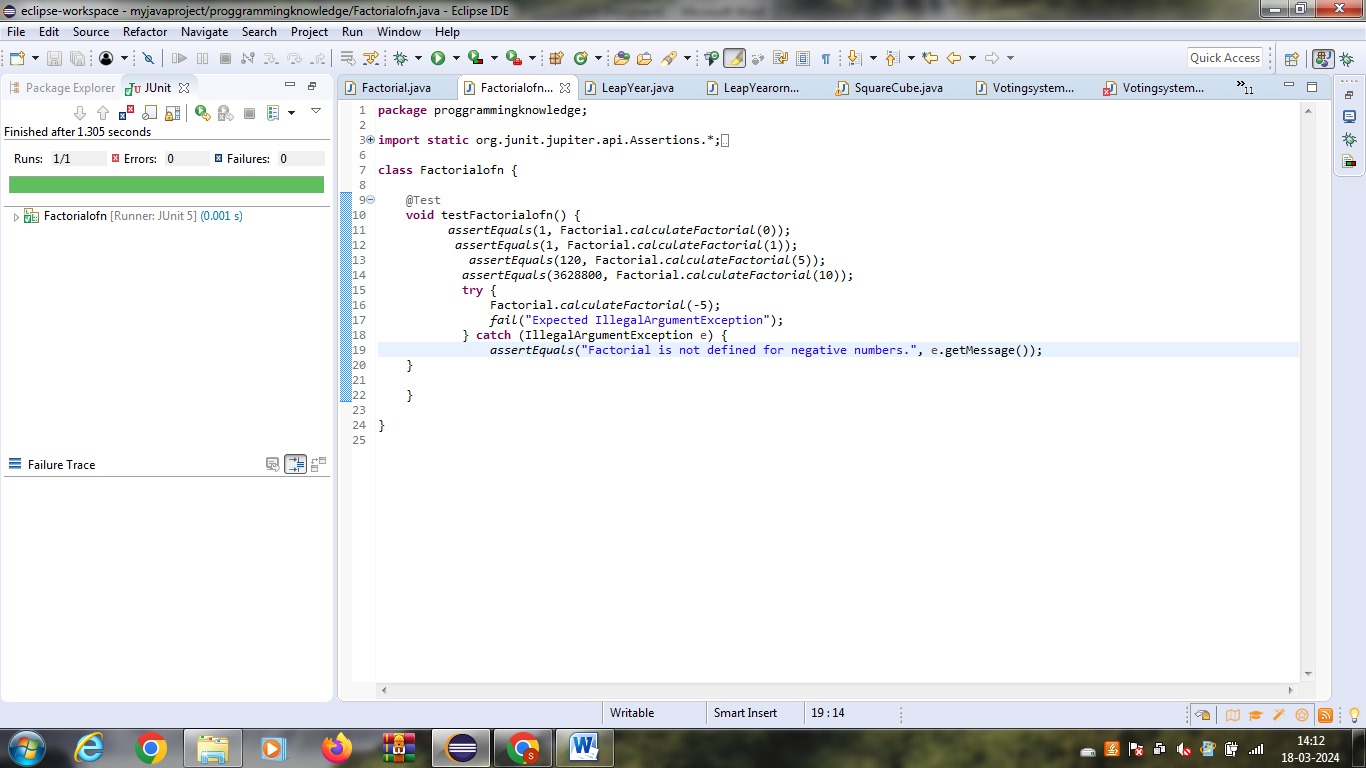
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 5 | 120 |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

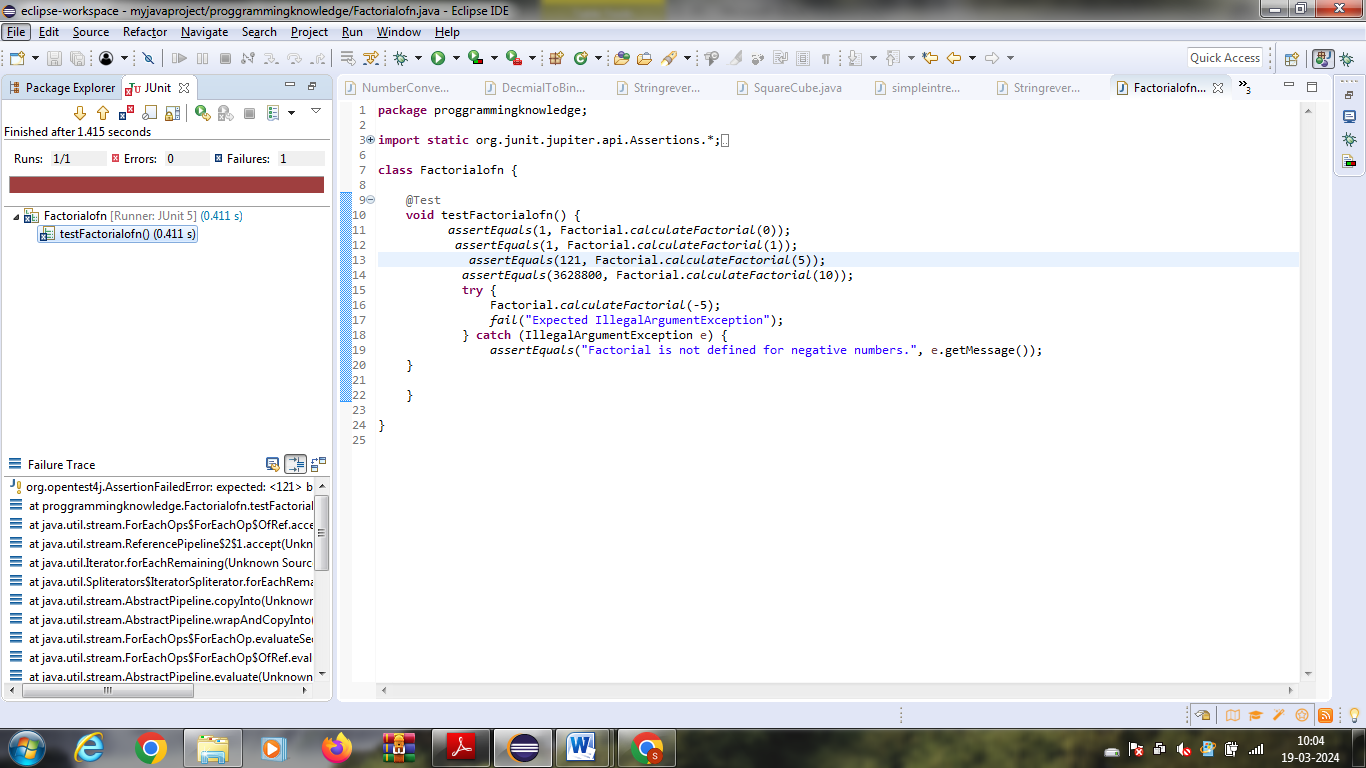
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 5 | 120 | 120 | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 5 | 121 | failure | FAILURE |



**EX NO:19**

**Find the year of the given date is leap year or not .The output values should verify using white box testing**

package proggrammingknowledge;

public class LeapYear {

public static String checkLeapYear(int year) {

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {

return year + " is a leap year.";

} else {

return year + " is not a leap year.";

}

}

public static void main(String[] args) {

int year = 2024;

System.out.println(checkLeapYear(year));

}

}

package proggrammingknowledge;

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

import org.junit.jupiter.api.Test;

class LeapYearornot {

@Test

void testLeapYearornot() {

assertEquals("2024 is a leap year.", LeapYear.checkLeapYear(2024));

assertEquals("2021 is not a leap year.", LeapYear.checkLeapYear(2021));

assertEquals("2000 is a leap year.", LeapYear.checkLeapYear(2000));

assertEquals("1900 is not a leap year.", LeapYear.checkLeapYear(1900));

}

}

**OUTPUT**

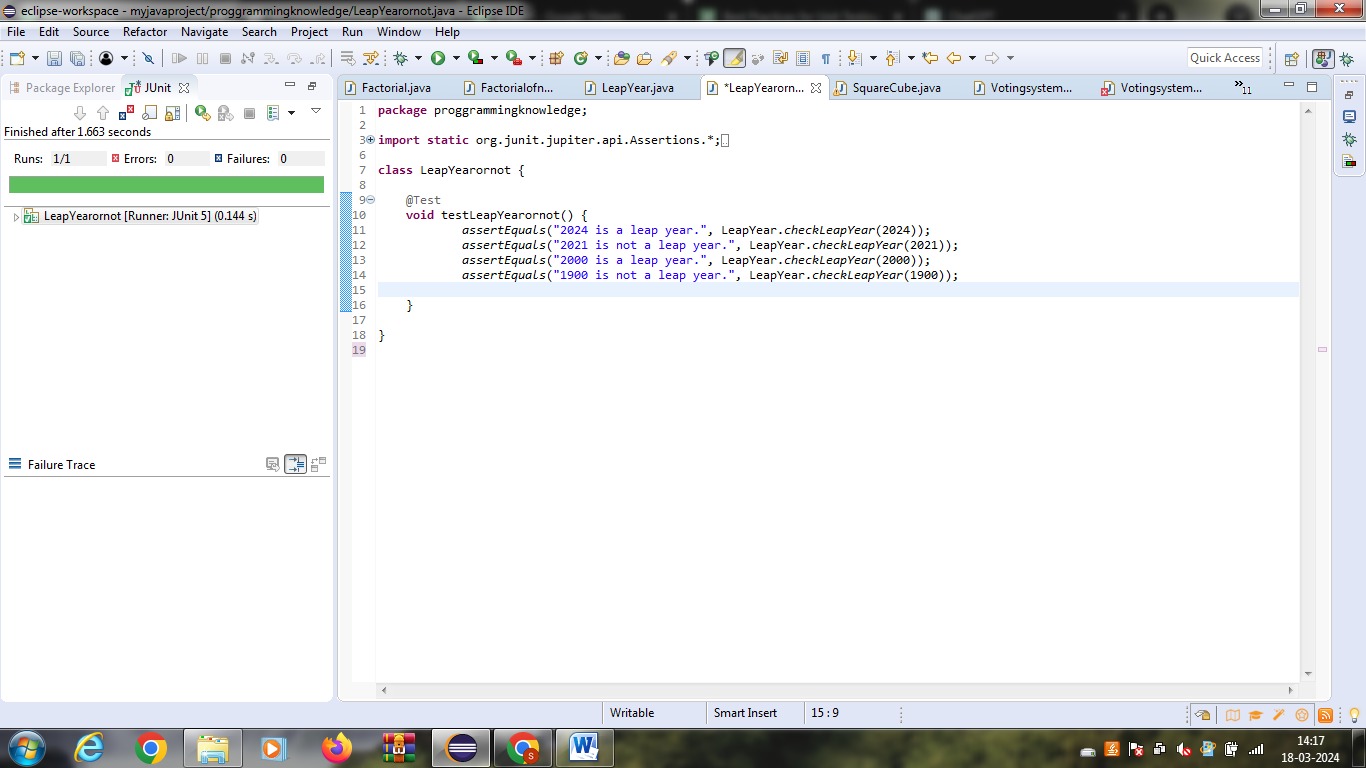
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 2024 | Leap year |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

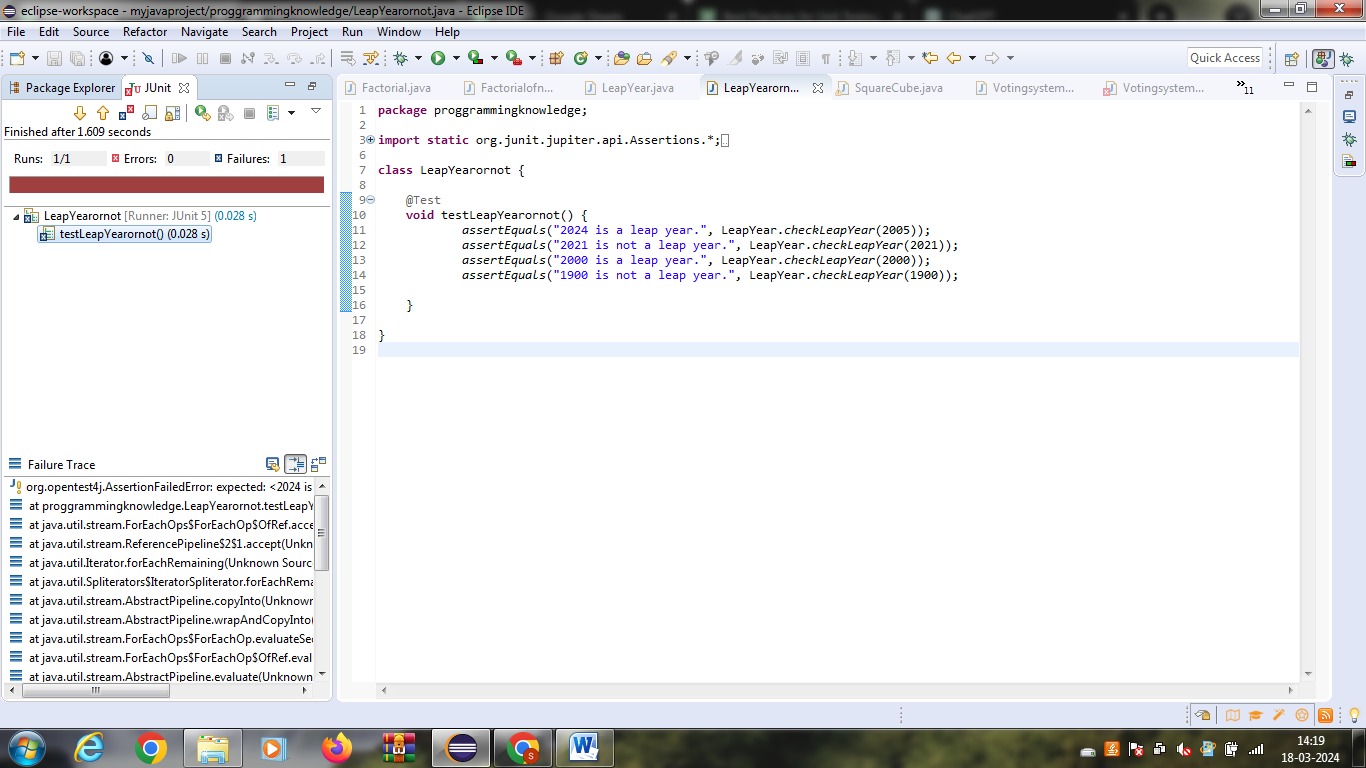
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 2024 | Leap year | Leap year | SUCCESS |



**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 2005 | Leap year | failure | FAILURE |



**EX NO:20**

**Write a program to find the square, cube of the given decimal number. The output values should verify using white box testing**

**package** proggrammingknowledge;

**import** java.util.Scanner;

**public** **class** SquareCube {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter a decimal number: ");

**double** number = scanner.nextDouble();

**double** square = number \* number;

**double** cube = number \* number \* number;

System.***out***.println("Square of the number: " + square);

System.***out***.println("Cube of the number: " + cube);

}

}

**package proggrammingknowledge;**

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

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

**class SquareCubeconvert {**

**@Test**

**void testSquareCubeconvert() {**

**double number = 2.5;**

**double square = number \* number;**

**double cube = number \* number \* number;**

**assertEquals(square, 6.25, 0.0001);**

**assertEquals(cube, 15.625, 0.0001);**

**}**

**}**

**OUTPUT**

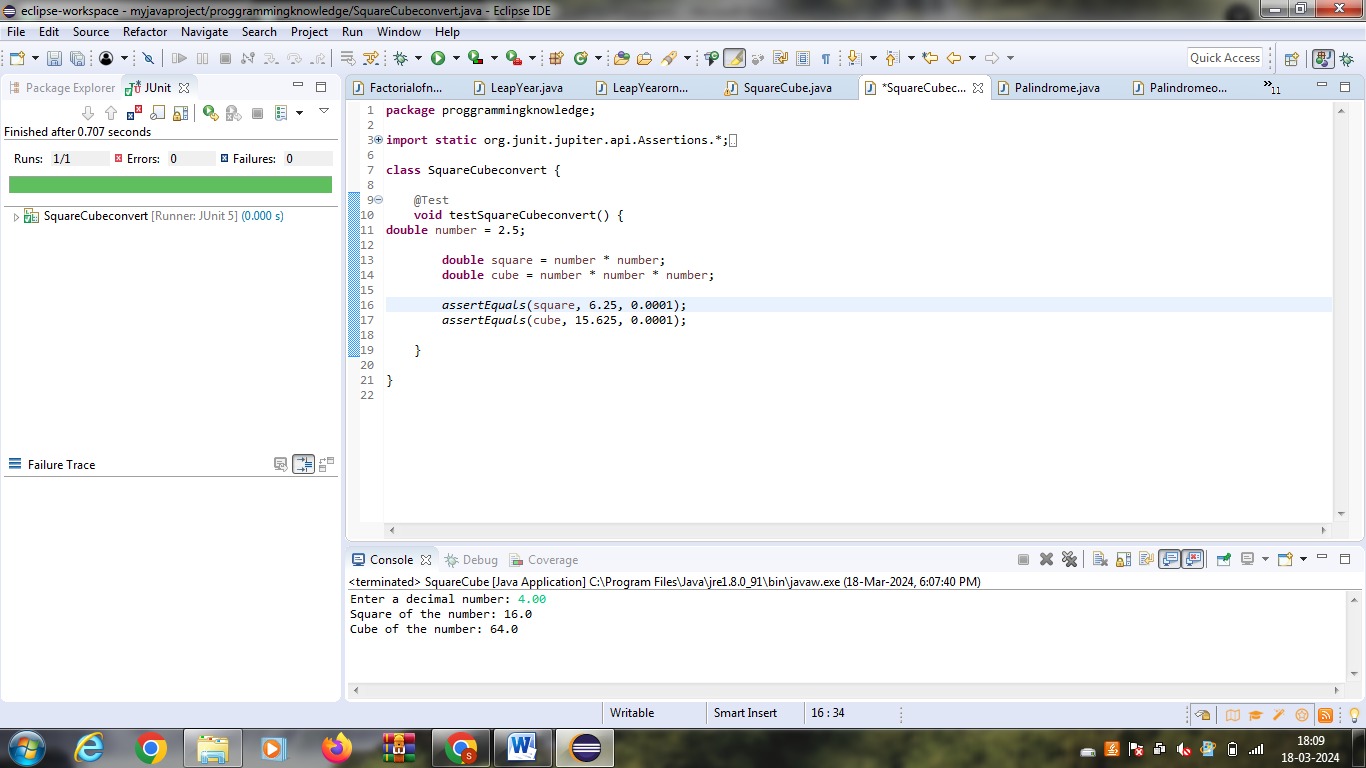
|  |  |
| --- | --- |
| **Input** | **Actual Output** |
| 2.5 | 6.25,0.0001 |

**TEST CASES:**

**Test Case 1:**

**Test Case Name:**

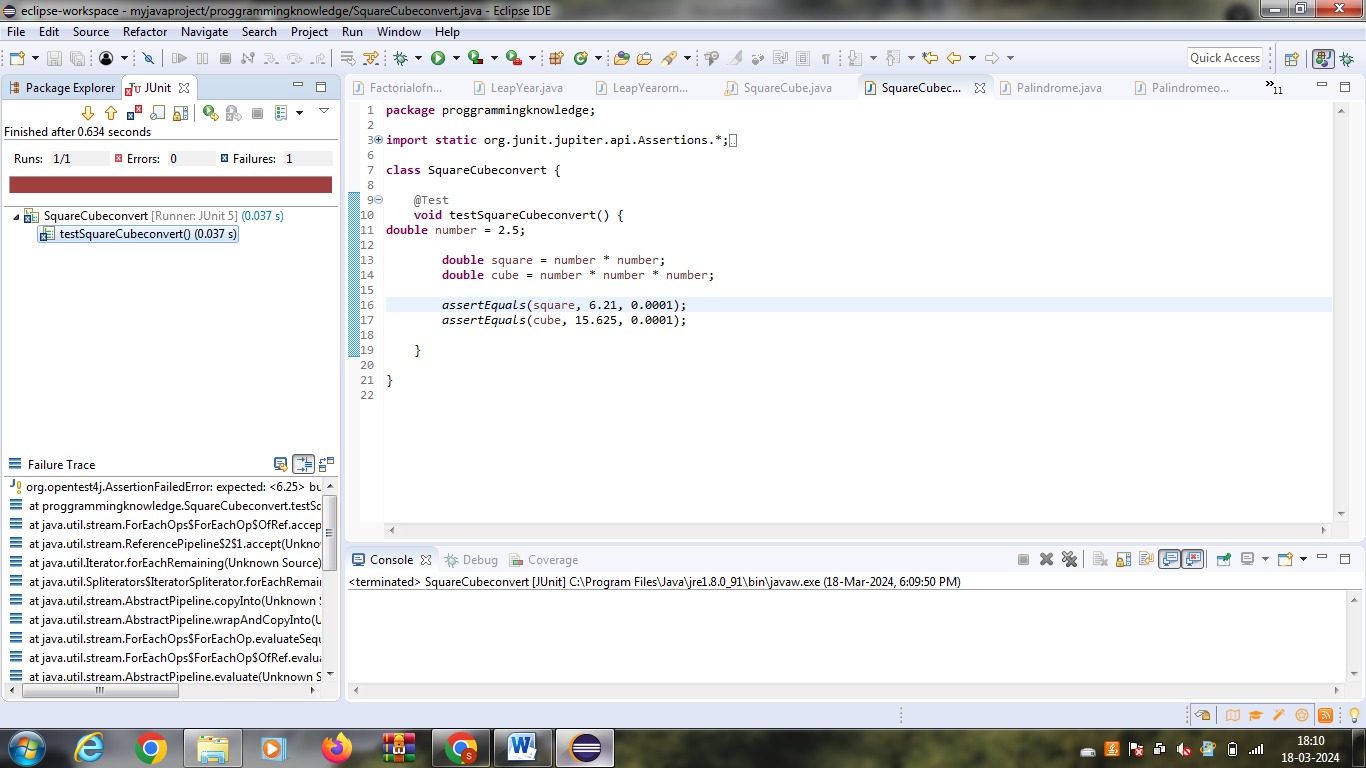
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 2.5 | 6.25,0.0001 | 6.25,0.0001 | SUCCESS |

****

**Test Case 2:**

**Test Case Name:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** | **Remarks** |
| 2.5 | 6.21,0.0001 | Failure | FAILURE |

****