**Week 2 : Advanced Junit Testing Exercise**

**Exercise 1 : Parameterized Tests**

**EvenChecker.java**

package com.example;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package com.example;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

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

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10, 0, -2})

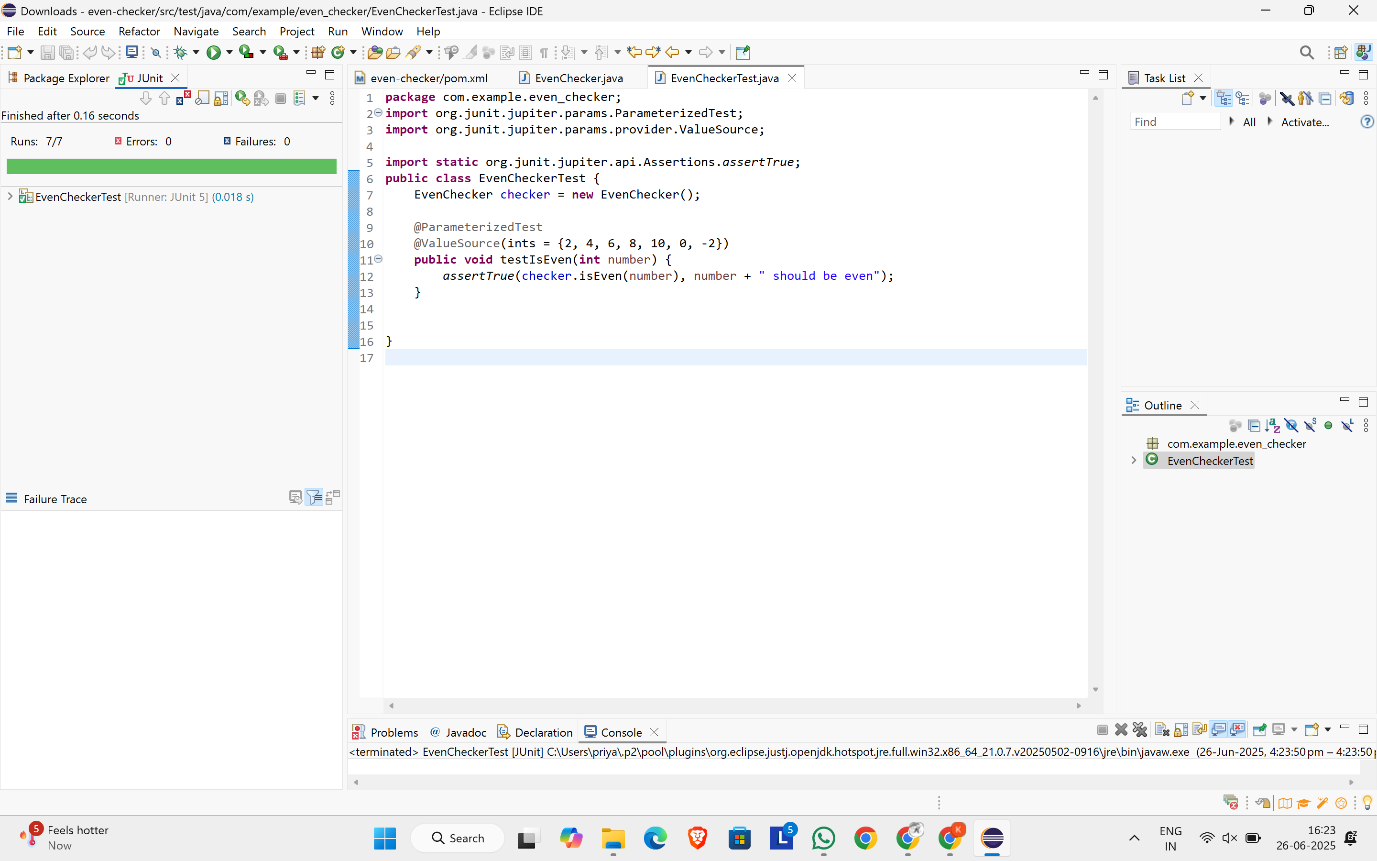
public void testIsEven(int number) {

assertTrue(checker.isEven(number), number + " should be even");

}

}

**OUTPUT**

****

**Exercise 2 : Test Suites and Categories**

**MathUtilsTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

@Test

void testAdd() {

assertEquals(5, 2 + 3);

}

}

**StringUtilsTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class StringUtilsTest {

@Test

void testUpperCase() {

assertEquals("HELLO", "hello".toUpperCase());

}

}

**AllTests.java**

package com.example;

import org.junit.platform.suite.api.SelectClasses;

import org.junit.platform.suite.api.Suite;

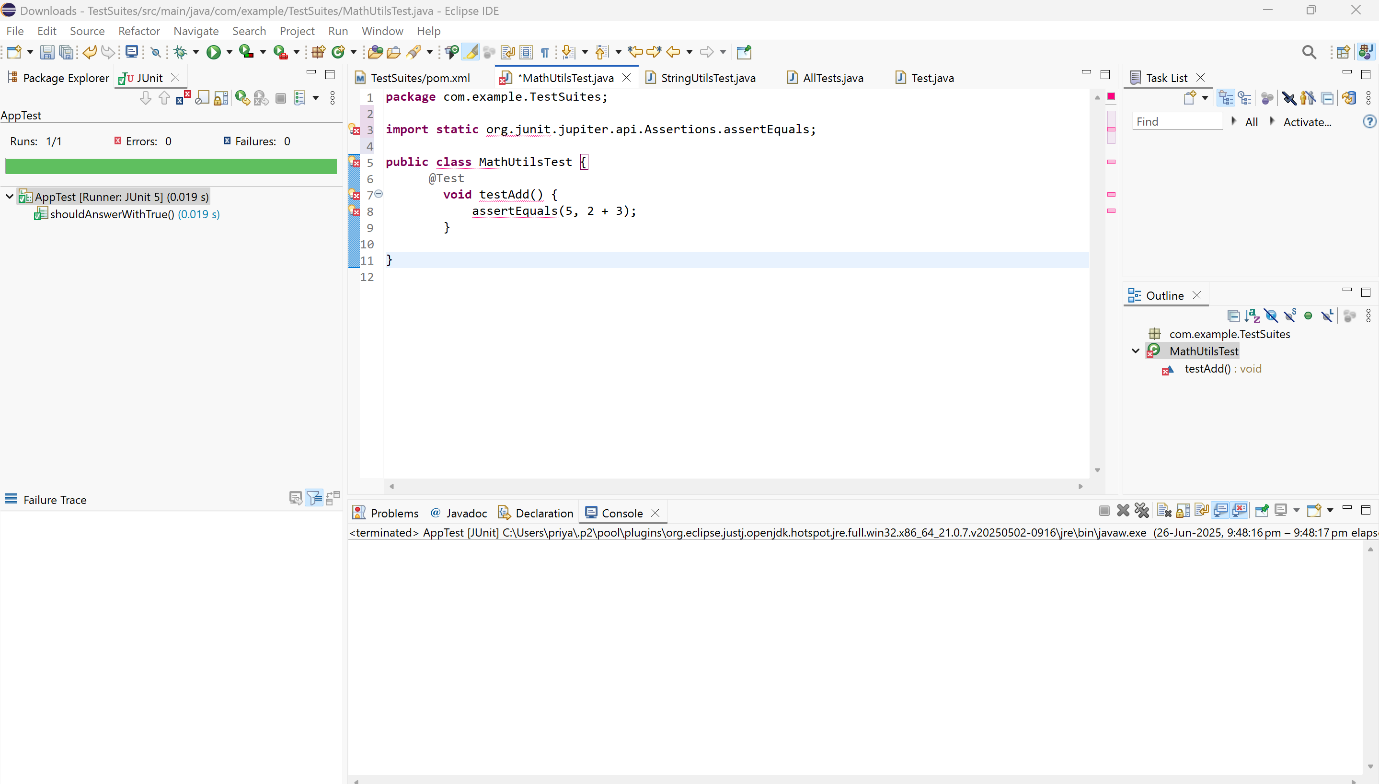
@Suite

@SelectClasses({ MathUtilsTest.class, StringUtilsTest.class })

public class AllTests {

}

**OUTPUT**



**Exercise 3 : Test Execution Order**

**OrderedTest.java**

package com.example;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

import org.junit.jupiter.api.MethodOrderer;

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(2)

void testSecond() {

System.out.println("Executing testSecond()");

assertTrue(true);

}

@Test

@Order(1)

void testFirst() {

System.out.println("Executing testFirst()");

assertTrue(true);

}

@Test

@Order(3)

void testThird() {

System.out.println("Executing testThird()");

assertTrue(true);

}}

**Pom.xml**

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

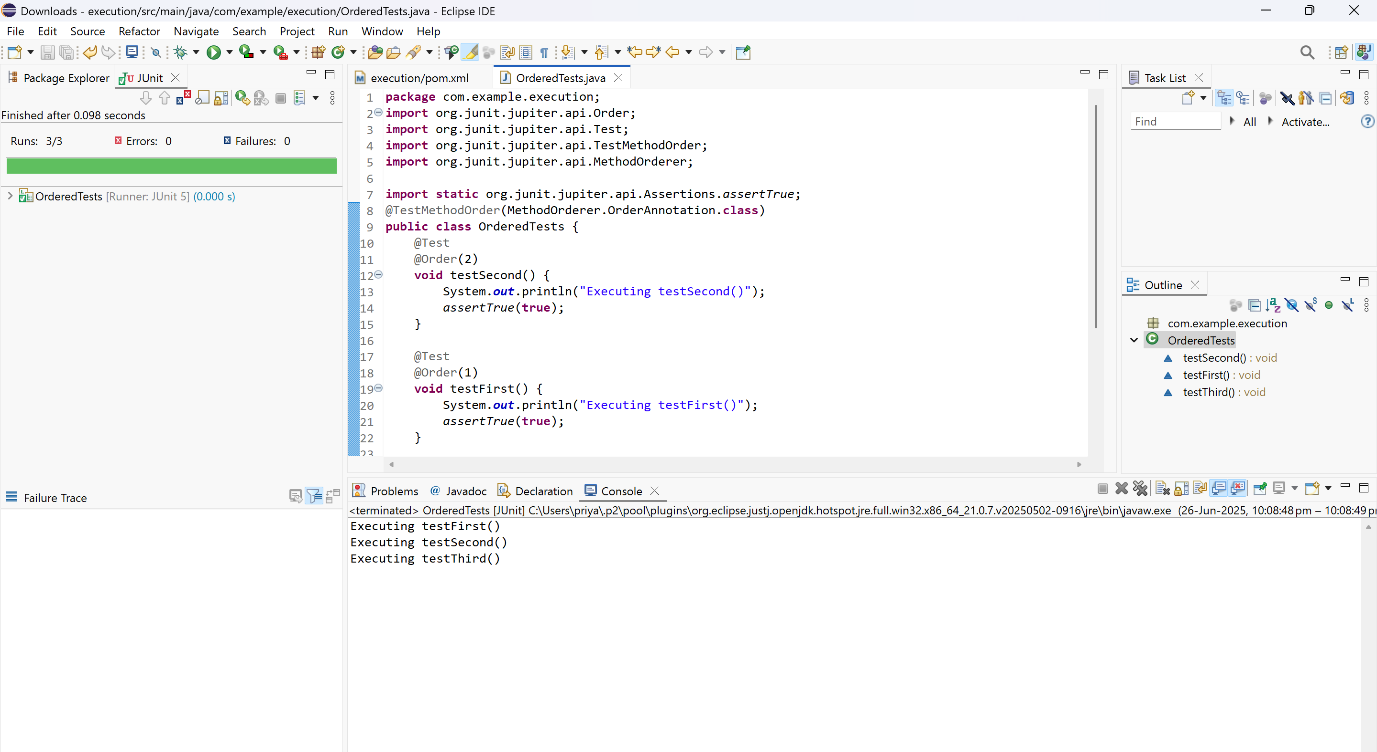
<version>3.0.0-M9</version>

</plugin>

</plugins>

</build>

**OUTPUT**

****

**Exercise 4 : Exception Testing**

**ExceptionThrower.java**

package com.example;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null || input.isEmpty()) {

throw new IllegalArgumentException("Input cannot be null or empty");

}

System.out.println("Valid input: " + input);

}

}

**ExceptionThrowerTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

void testThrowException() {

ExceptionThrower thrower = new ExceptionThrower();

assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException("");

});

assertThrows(IllegalArgumentException.class, () -> {

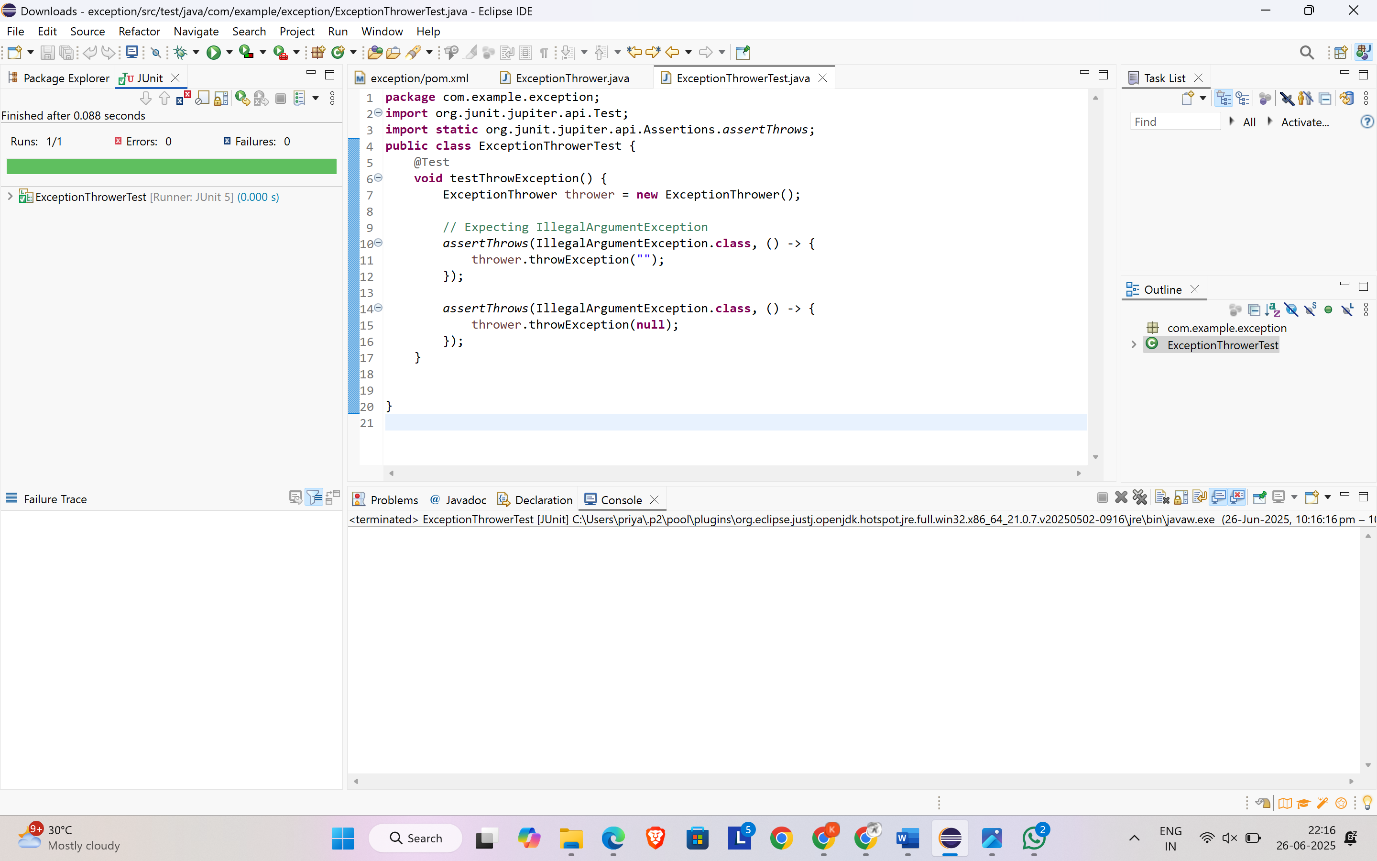
thrower.throwException(null);

});

}

}

**OUTPUT**



**Exercise 5 : Timeout and Performance Testing**

**PerformanceTester.java**

package com.example;

public class PerformanceTester {

public void performTask() {

try {

Thread.sleep(400);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**PerformanceTesterTest.java**

package com.example;

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

@Test

void testPerformTaskCompletesWithin500ms() {

PerformanceTester tester = new PerformanceTester();

assertTimeout(Duration.ofMillis(500), () -> {

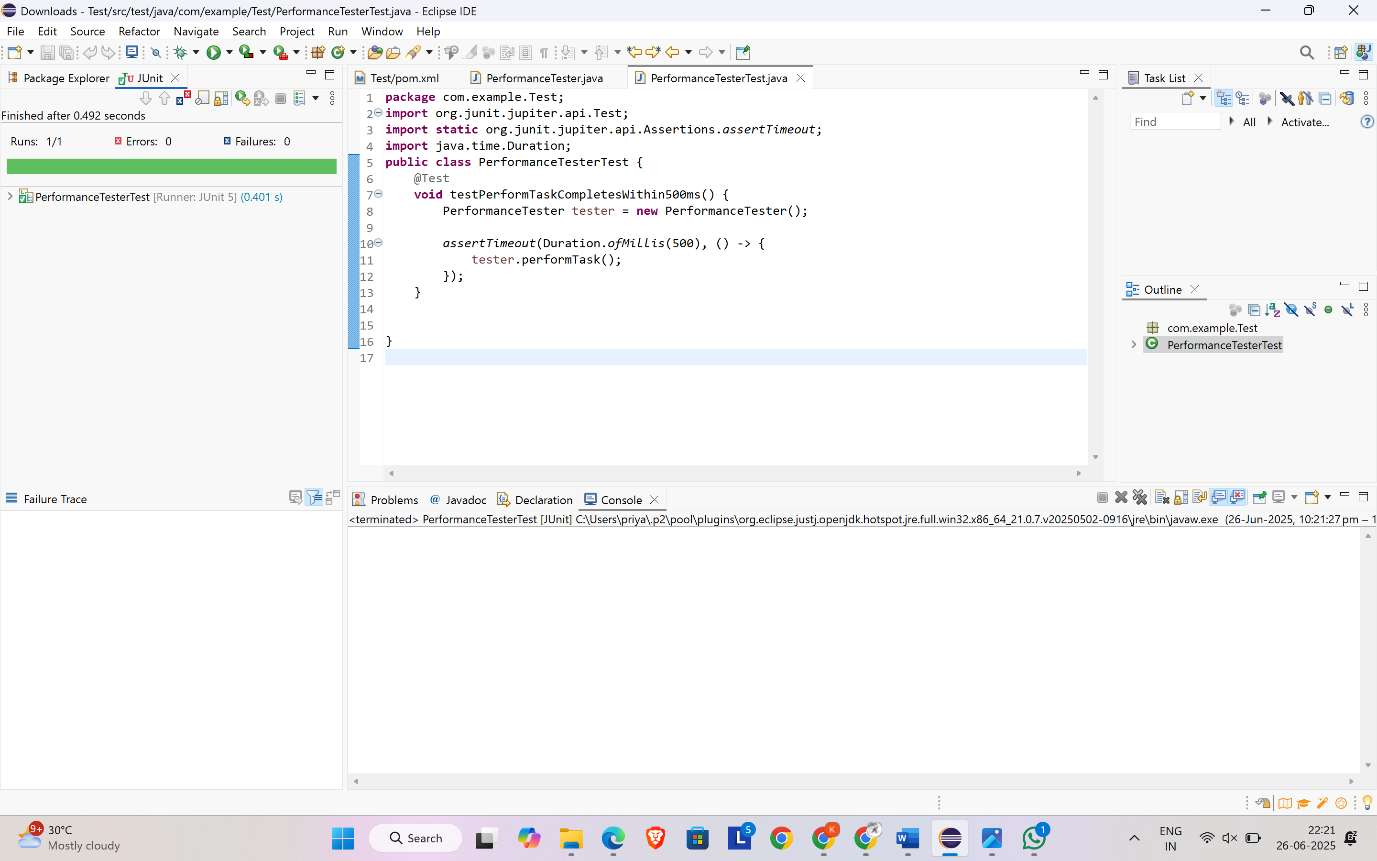
tester.performTask();

});

}

}

**OUTPUT**

****