**WEEK-2\_ADVANCED JUNIT TESTING EXERCISES**

**SUPERSETID:6393676**

**EXERCISE 1:**

Parameterized Tests Scenario: You want to test a method that checks if a number is even. Instead of writing multiple test cases, you will use parameterized tests to run the same test with different inputs.

**SOLUTION:**

**EvenChecker.java**

package demo.basic;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package demo.basic;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

@ParameterizedTest

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

public void testIsEvenWithEvenNumbers(int number) {

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

}

@ParameterizedTest

@ValueSource(ints = {1, 3, 5, 7, 9})

public void testIsEvenWithOddNumbers(int number) {

assertFalse(checker.isEven(number), number + " should be odd");

}

}

**Pom.xml**

<dependency>

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

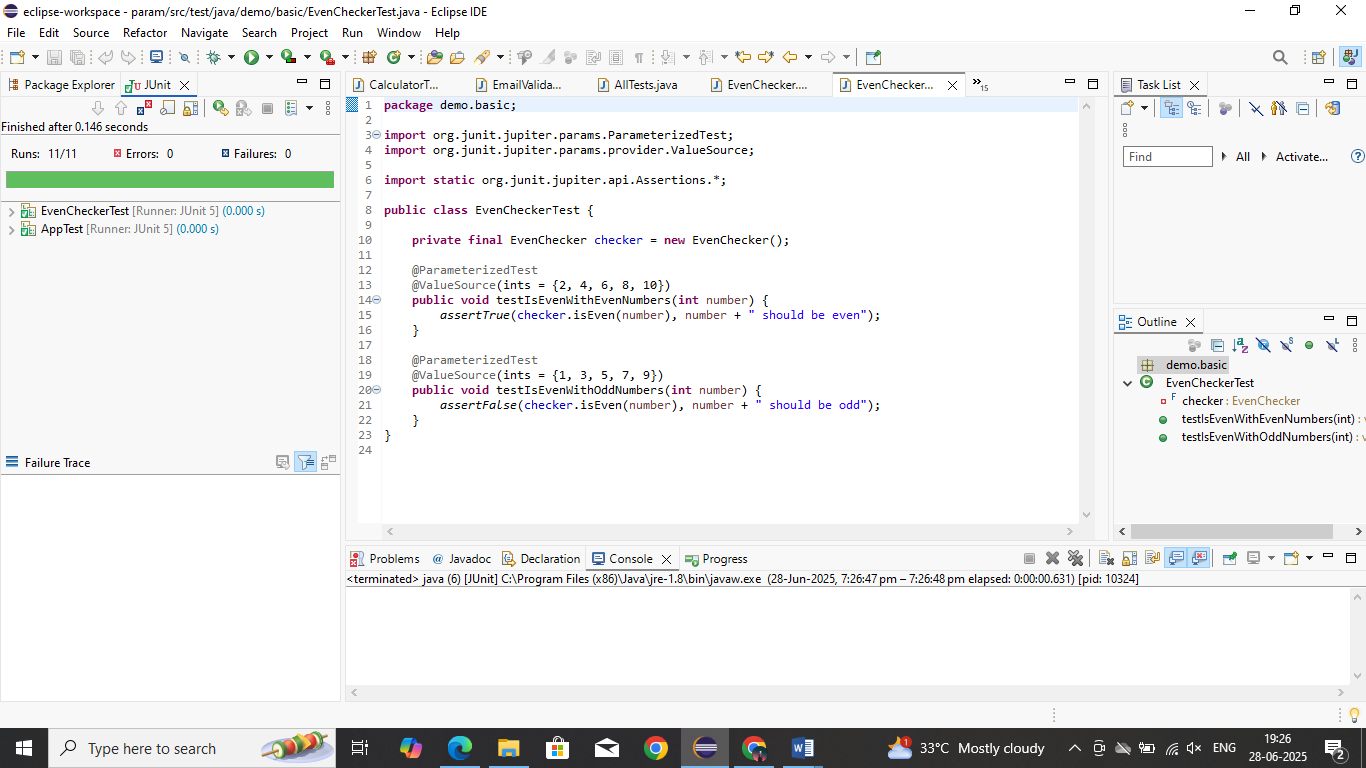
<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

**OUTPUT:**



**EXERCISE 2:**

Test Suites and Categories Scenario: You want to group related tests into a test suite and categorize them.

**SOLUTION:**

**CalculatorTest.java**

package demo.basic;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAddition() {

assertEquals(5, 2 + 3);

}

}

**EmailValidatorTest.java**

package demo.basic;

import org.junit.jupiter.api.Test;

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

public class EmailValidatorTest {

@Test

void testEmailFormat() {

assertTrue("test@example.com".contains("@"));

}

}

**AllTests.java**

package demo.basic;

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

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

@Suite

@SelectClasses({

CalculatorTest.class,

EmailValidatorTest.class

})

public class AllTests {

// No code needed – annotations handle everything

}

**Pom.xml**

<dependency>

<groupId>org.junit.platform</groupId>

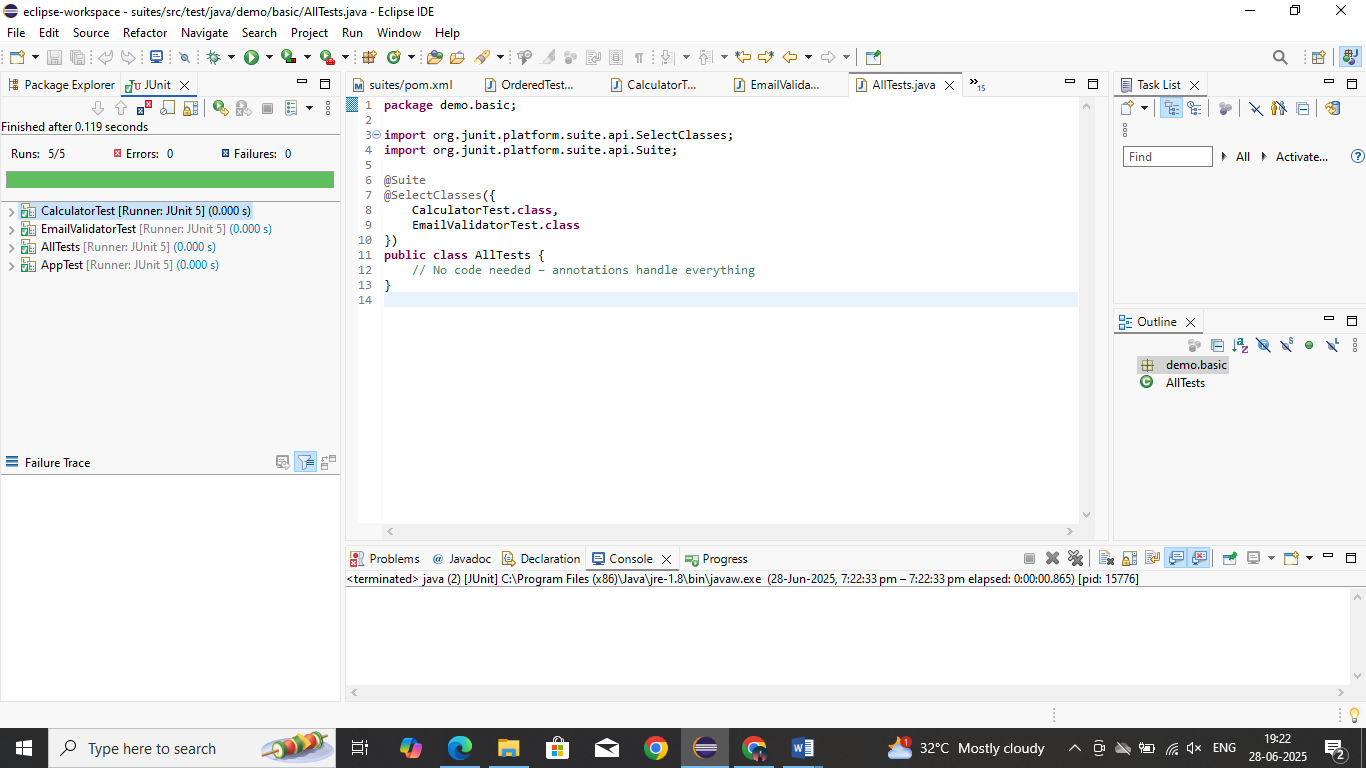
<artifactId>junit-platform-suite</artifactId>

<version>1.10.0</version>

<scope>test</scope>

</dependency>

**OUTPUT:**



**EXERCISE 3:**

Test Execution Order Scenario: You want to control the order in which tests are executed.

**SOLUTION:**

**OrderedTest.java**

package demo.order;

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

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

static StringBuilder sequence = new StringBuilder();

@Test

@Order(1)

void testStart() {

sequence.append("A");

System.out.println("testStart executed");

assertEquals("A", sequence.toString());

}

@Test

@Order(3)

void testEnd() {

sequence.append("C");

System.out.println("testEnd executed");

assertEquals("ABC", sequence.toString());

}

@Test

@Order(2)

void testMiddle() {

sequence.append("B");

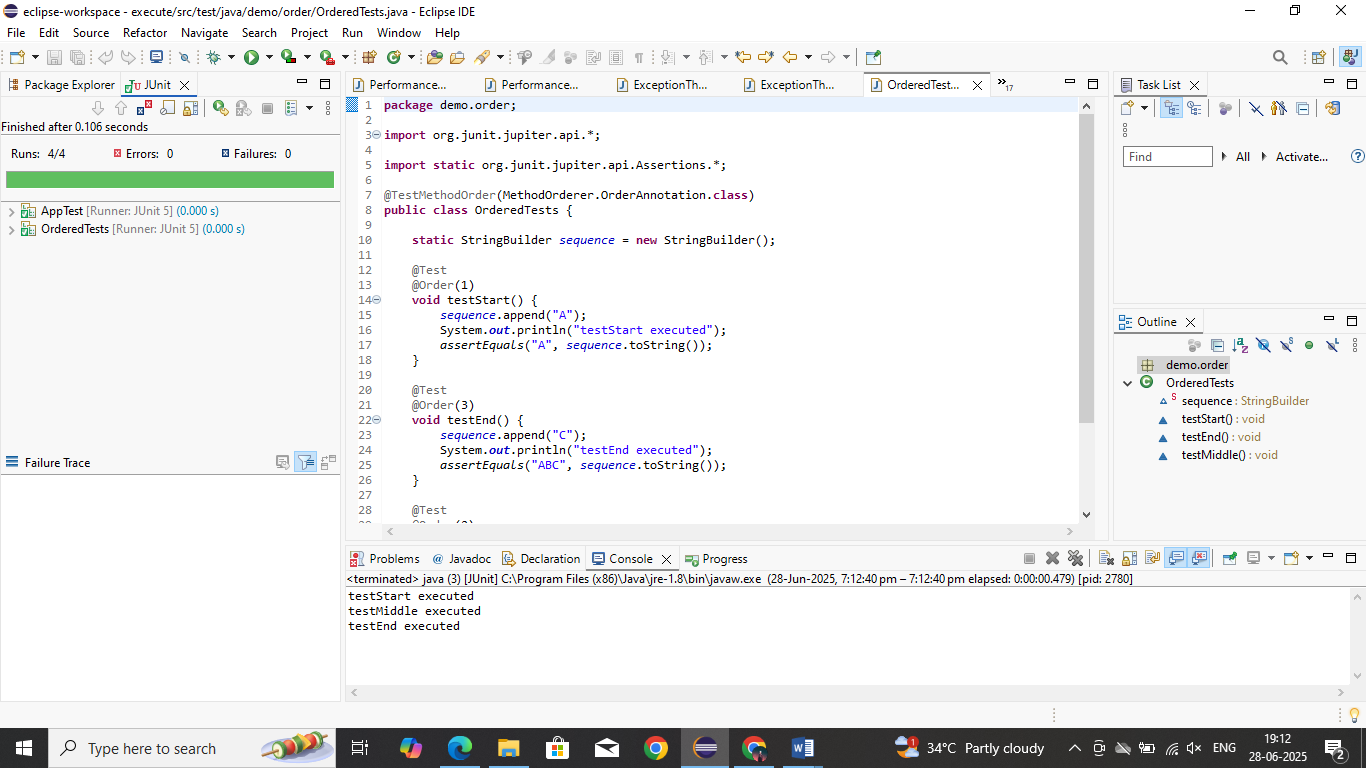
System.out.println("testMiddle executed");

assertEquals("AB", sequence.toString());

}

}

**OUTPUT:**



**EXERCISE 4:**

Exception Testing Scenario: You want to test that a method throws the expected exception.

**SOLUTION:**

**ExceptionThrower.java**

package demo.exceptions;

public class ExceptionThrower {

public void throwException(String input) {

if (input == null) {

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

}

// Normal logic would go here

}

}}

**ExceptionThrowerTest.java**

package demo.exceptions;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

@Test

void testThrowExceptionWithNull() {

ExceptionThrower thrower = new ExceptionThrower();

IllegalArgumentException exception = assertThrows(

IllegalArgumentException.class,

() -> thrower.throwException(null),

"Expected IllegalArgumentException for null input"

);

assertEquals("Input cannot be null", exception.getMessage());

}

@Test

void testThrowExceptionWithNonNull() {

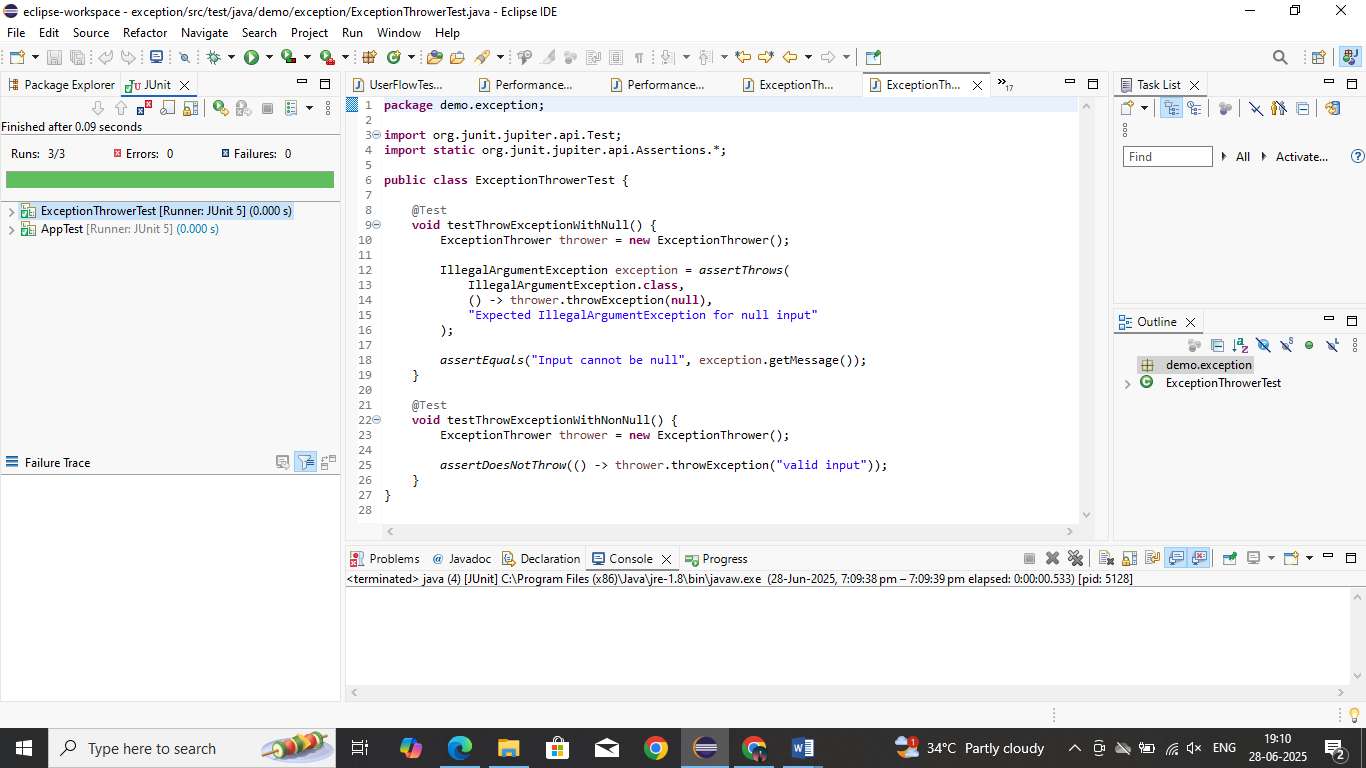
ExceptionThrower thrower = new ExceptionThrower();

assertDoesNotThrow(() -> thrower.throwException("valid input"));

}

}

**OUTPUT:**



**EXERCISE 5:**

Timeout and Performance Testing Scenario: You want to ensure that a method completes within a specified time limit.

**SOLUTION:**

**PerforamnceTester.java**

package demo.performance;

public class PerformanceTester {

public void performTask() {

// Simulate a task that takes time

try {

Thread.sleep(400); // 400 milliseconds

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**PerformanceTesterTest.java**

package demo.performance;

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

@Test

void testPerformTaskCompletesInTime() {

PerformanceTester tester = new PerformanceTester();

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

tester.performTask();

});

}

}

**OUTPUT:**

