**WEEK 2 : JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

**Scenario:**

**You need to set up JUnit in your Java project to start writing unit tests.**

**Code:**

**Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

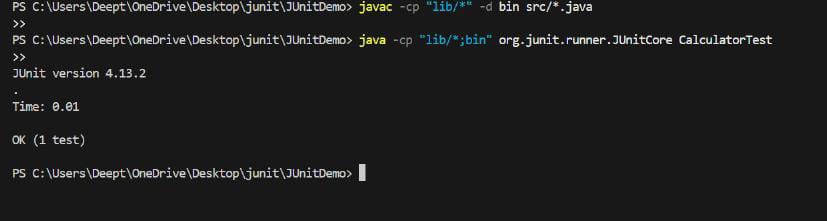
Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}

**Output:**



**Exercise 2:** Writing Basic JUnit Tests

**Scenario:**

You need to write basic JUnit tests for a simple Java class.

**Code:**

**Calculator.java**

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;

    }

}

**CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

    Calculator calc = new Calculator();

    @Test

    public void testAdd() {

        assertEquals(5, calc.add(2, 3));

    }

    @Test

    public void testSubtract() {

        assertEquals(2, calc.subtract(5, 3));

    }

    @Test

    public void testMultiply() {

        assertEquals(15, calc.multiply(5, 3));

    }

    @Test

    public void testDivide() {

        assertEquals(2, calc.divide(6, 3));

    }

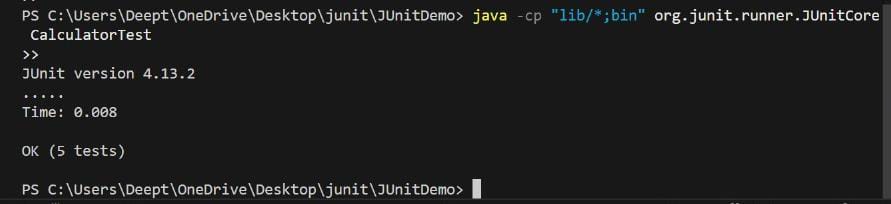
    @Test(expected = IllegalArgumentException.class)

    public void testDivideByZero() {

        calc.divide(5, 0);

    } }

**Output:**



**Exercise 3:** Assertions in JUnit

**Scenario:** You need to use different assertions in JUnit to validate your test results.

**Code:**

**AssertionsTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

    @Test

    public void testAssertions() {

        assertEquals(5, 2 + 3);

        assertTrue(5 > 3);

        assertFalse(5 < 3);

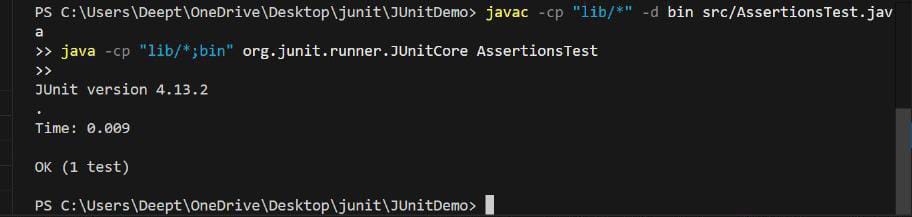
        assertNull(null);

        assertNotNull(new Object());

    }

}

**Output:**



**Exercise 4:** Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

**Scenario:**

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

**Code:**

**CalculatorAaaTest.java**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAaaTest {

    private Calculator calc;

    @Before

    public void setUp() {

        System.out.println("Setting up Calculator...");

        calc = new Calculator();

    }

    @Test

    public void testAdd() {

        int result = calc.add(2, 3);

        assertEquals(5, result);

    }

    @Test

    public void testSubtract() {

        int result = calc.subtract(5, 2);

        assertEquals(3, result);

    }

    @After

    public void tearDown() {

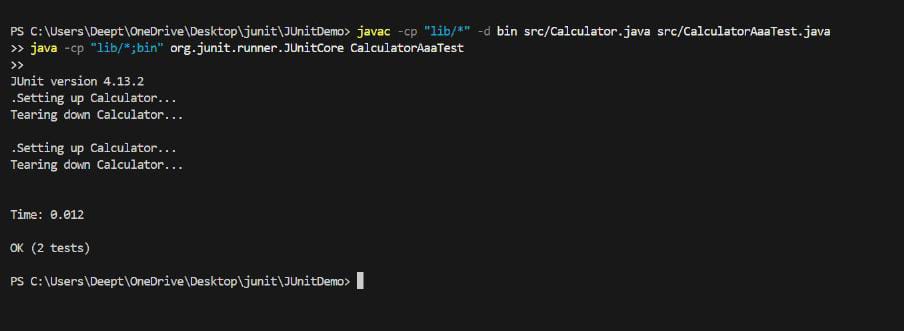
        System.out.println("Tearing down Calculator...\n");

        calc = null;

    }

}

**Output:**



**2) Advanced JUnit Testing Exercises**

**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.

**Code:**

**EvenChecker.java**

public class EvenChecker {

    public boolean isEven(int number) {

        return number % 2 == 0;

    }

}

**EvenCheckerText.java**

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

    EvenChecker checker = new EvenChecker();

    @ParameterizedTest

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

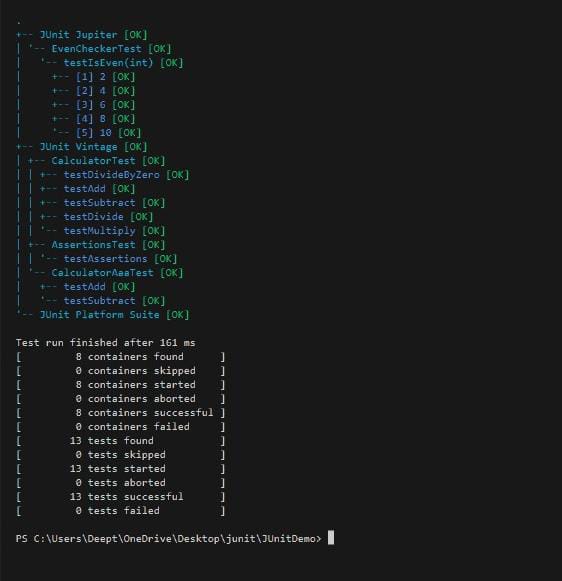
    public void testIsEven(int number) {

        assertTrue(checker.isEven(number));

    }

}

**Output:**



**Exercise 2:** Test Suites and Categories

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

**Code:**

**NumberSignTest.java**

import org.junit.jupiter.api.Test;

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

public class NumberSignTest {

    @Test

    public void testPositive() {

        assertTrue(10 > 0);

    }

    @Test

    public void testNegative() {

        assertTrue(-5 < 0);

    }

}

**OddCheckerTest.java**

import org.junit.jupiter.api.Test;

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

public class OddCheckerTest {

    @Test

    public void testIsOdd() {

        int number = 7;

        assertTrue(number % 2 != 0);

    }

    @Test

    public void testIsNotOdd() {

        int number = 4;

        assertFalse(number % 2 != 0);

    }

}

**AllTests.java**

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

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

@Suite

@SelectClasses({

    NumberSignTest.class,

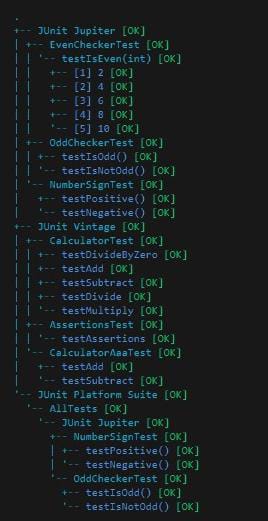
    OddCheckerTest.class

})

public class AllTests {

}

**Output:**



**Exercise 3:** Test Execution Order

**Scenario:** You want to control the order in which tests are executed.

**Code:**

**OrderedTests.java**

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.TestMethodOrder;

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

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

@TestMethodOrder(OrderAnnotation.class)

public class OrderedTests {

    @Test

    @Order(3)

    public void testC() {

        System.out.println("Running testC (Order 3)");

        assertTrue(true);

    }

    @Test

    @Order(1)

    public void testA() {

        System.out.println("Running testA (Order 1)");

        assertTrue(true);

    }

    @Test

    @Order(2)

    public void testB() {

        System.out.println("Running testB (Order 2)");

        assertTrue(true);

    }

}

**Output:**



**Exercise 4:** Exception Testing

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

**Code:**

**ExceptionThrower.java**

public class ExceptionThrower {

    public void throwException(boolean shouldThrow) {

        if (shouldThrow) {

            throw new IllegalArgumentException("Exception was triggered");

        }

    }

}

**ExceptionThrowerTest.java**

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

    ExceptionThrower thrower = new ExceptionThrower();

    @Test

    public void testThrowsException() {

        assertThrows(IllegalArgumentException.class, () -> {

            thrower.throwException(true);

        });

    }

    @Test

    public void testDoesNotThrowException() {

        assertDoesNotThrow(() -> {

            thrower.throwException(false);

        });

    }

}

**Output:**



**Exercise 5:** Timeout and Performance Testing

**Scenario:** You want to ensure that a method completes within a specified time limit**.**

**Code:**

**PerformanceTester.java**

public class PerformanceTester {

    public void performTask() {

        try {

            Thread.sleep(1000); // 1 second

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

    }

}

**PerformanceTesterTest.java**

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

    PerformanceTester tester = new PerformanceTester();

    @Test

    public void testPerformTaskWithinTimeout() {

        assertTimeout(Duration.ofSeconds(2), () -> {

            tester.performTask();

        });

    }

}

**Output:**

