**JAVA TESTING IN VISUAL STUDIO CODE**

# **Objective**

By the end of this lab, students will:

1. Set up a Java testing environment in VS Code.
2. Write unit tests using JUnit.
3. Execute and debug tests.
4. Understand test results and improve code accordingly.

# **Tutorial - How to run JUnit Test in VSCode**

1. Install the “Java Extension Pack” extension in VSCode. Be sure the one by Microsoft.
2. Install “Test Runner for Java” extension in VSCode.
3. Create a new Java project and keep the program directory like this:

A screenshot of a computer

Description automatically generated

1. Check the “settings.json” file.

A screenshot of a computer code

Description automatically generated

1. In Activity Bar, select “Testing” panel and click “Enable Java Tests” button, then select “JUnit” option as the test framework.

A screenshot of a computer

Description automatically generated

A blue and white rectangular object

Description automatically generated

1. In “src/main” folder, create a class with methods. Take the following “Calculator” class as an example.

package main;

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) throws IllegalArgumentException {

        if (b == 0) {

            throw new IllegalArgumentException("Division by zero is not allowed.");

        }

        return a / b;

    }

}

1. In “src/test” folder, create a file to test the above methods.

package test;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertThrows;

import org.junit.Test;

import main.Calculator;

public class CalculatorTest {

    Calculator calculator = new Calculator();

    @Test

    public void testDivideValidInputs() {

        // Test normal division

        assertEquals(2, calculator.divide(10, 5));

        assertEquals(0, calculator.divide(1, 2));

        assertEquals(-3, calculator.divide(-9, 3));

    }

    @Test

    public void testDivideByZero() {

        // Test division by zero

        Exception exception = assertThrows(IllegalArgumentException.class, () -> {

            calculator.divide(10, 0);

        });

        assertEquals("Division by zero is not allowed.", exception.getMessage());

    }

    @Test

    public void testAdd() {

        assertEquals(10, calculator.add(7, 3));

        assertEquals(-2, calculator.add(-5, 3));

        assertEquals(-20, calculator.add(-5, -15));

    }

    @Test

    public void testSubtract() {

        assertEquals(2, calculator.subtract(7, 5));

        assertEquals(-3, calculator.subtract(-5, 2));

        assertEquals(-20, calculator.subtract(-5, 15));

    }

    @Test

    public void testMultiply() {

        assertEquals(12, calculator.multiply(4, 3));

        assertEquals(-6, calculator.multiply(-3, 2));

        assertEquals(24, calculator.multiply(-6, -4));

    }

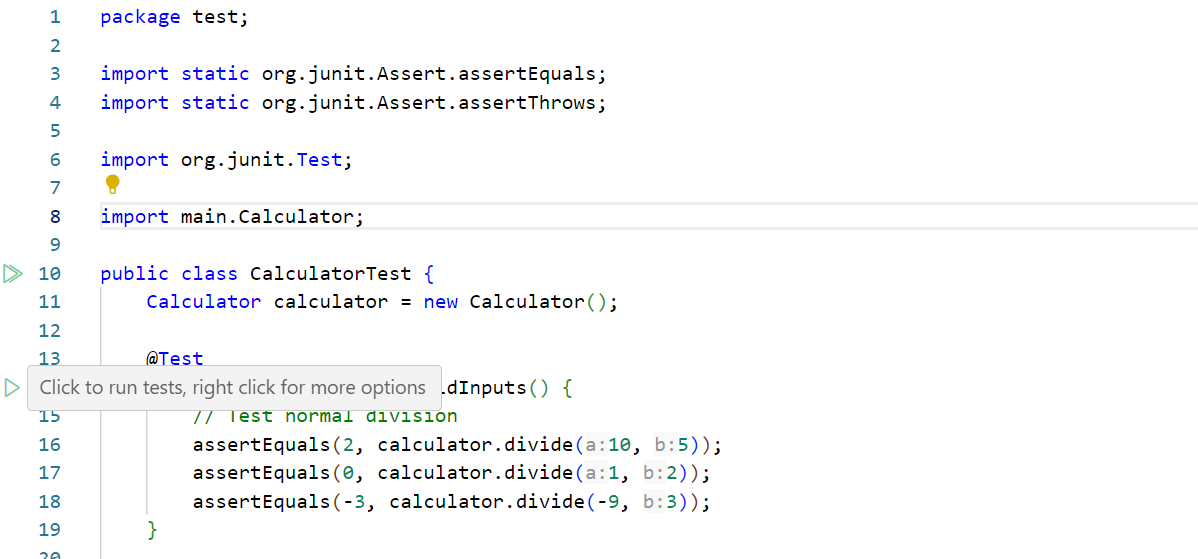
}

***Explanations***:

* + **@Test**: tells JUnit that the public void method to which it is attached can be run as a test case.
  + **assertEquals(expected result, actual result)**: asserts that two results are equal. If they are not, an [AssertionError](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is thrown.
  + **assertThrows(expected type of the exception, function that is expected to throw an exception when executed)**: asserts that runnable throws an exception of type expectedThrowable when executed. If it does, the exception object is returned. If it does not throw an exception, an [AssertionError](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) is thrown. If it throws the wrong type of exception, an AssertionError is thrown describing the mismatch; the exception that was actually thrown can be obtained by calling [AssertionError.getCause](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).

1. Run tests

Run each test



Run all tests

A screenshot of a computer program

Description automatically generated

Run tests with coverage

A screenshot of a computer program

Description automatically generated

***Results***:

The following capture notices that the program passed 4/5 tests.

A screenshot of a computer

Description automatically generated

* The green ticks mean passed tests
* The red crosses mean failed tests. You need to check your program or your test in those cases.

A screenshot of a computer

Description automatically generated

Test Coverage shows the tests with their coverage percentage. A color indicator also gives a visual cue about the coverage percentage. Hover over each line to see more details about the coverage results.

1. Debug tests in JUnit Test are like Debug Java.

After debuging a failure test, you can know why the test is failed.

# **Assignment**

1. Open VSCode, create a new Java project named “YourFullName\_JunitTestLab”.
2. In “src/main” folder, create “StringManipulator” class and implement all following methods:

package main;

public class StringManipulator {

    // input: ExAmPLE => output: ELPmAxE

    public String reverse(String input) {

        // Your code here

        return null;

    }

    // input: example => output: false

    // input: ELpmpLE => output: true

    public boolean isPalindrome(String input) {

        // Your code here

        return true;

    }

    // input: ExAmPLE, part: Ex => output: true

    // input: ExAmPLE, part: ex => output: false

    public boolean isContained(String input, String part) {

        // Your code here

        return true;

    }

    // input: ExAm1PLE2 => output: ExAmPLE

    public String numberRemoved(String input) {

        // Your code here

        return null;

    }

    // input: ExAmPLE => output: EXAMPLE

    public String upperString(String input) {

        // Your code here

        return null;

    }

    // input: ExAmPLE => output: example

    public String lowerString(String input) {

        // Your code here

        return null;

    }

    // input: example string => output: Example String

    public String capitalString(String input) {

        // Your code here

        return null;

    }

    // input: Example => output: 3

    public int countVowels(String input) {

        // Your code here

        return 0;

    }

    // input: Hello, how are you? => output: 4

    public int countWords(String sentence) {

        // Your code here

        return 0;

    }

    // Input: input = "Java programming", substring = "programming" => Output: 5

    // Input: input = "Java programming", substring = "Programming" => Output: -1

    public int findSubstringIndex(String input, String substring) {

        // Your code here

        return -1;

    }

    // Input: "Hello World How Are You" => Output: "You Are How World Hello"

    public String reverseWords(String sentence) {

        // Your code here

        return null;

    }

}

1. In “src/test” folder, create “StringManipulatorTest” class to test all methods in “StringManipulator” class (at least all sample cases).

package test;

import org.junit.Test;

import main.StringManipulator;

public class StringManipulatorTest {

    StringManipulator stringManipulator = new StringManipulator();

@Test

    public void testReverse() {

       // Your code here

    }

    @Test

    public void testIsPalindrome() {

       // Your code here

    }

    @Test

    public void testIsContained() {

       // Your code here

    }

    @Test

    public void testNumberRemoved() {

       // Your code here

    }

    @Test

    public void testUpperString() {

       // Your code here

    }

    @Test

    public void testLowerString() {

       // Your code here

    }

    @Test

    public void testCapitalString() {

       // Your code here

    }

    @Test

    public void testCountVowels() {

       // Your code here

    }

    @Test

    public void testCountWords() {

       // Your code here

    }

    @Test

    public void testFindSubstringIndex() {

       // Your code here

    }

    @Test

    public void testReverseWords() {

       // Your code here

    }

}

1. Run test with coverage to make sure that your program passes all tests and cover all cases.
2. Add following tests with “timeout” parameter. Optionally specify timeout in milliseconds to cause a test method to fail if it takes longer than that number of milliseconds.

    @Test(timeout = 1000)

        public void testReverseWithTimeout() {

       // Your code here

    }

    @Test(timeout = 1000)

    public void testIsPalindromeWithTimeout() {

       // Your code here

    }

    @Test(timeout = 1000)

    public void testIsContainedWithTimeout() {

       // Your code here

    }

    @Test(timeout = 1000)

    public void testNumberRemovedWithTimeout() {

       // Your code here

    }

1. Push the project to GitHub. ***(You can use the README.md file to save the link to GitHub repository of the project)***

# **Submission**

A ZIP file contains the folder of the above project with the link in README.md.

File name: YourFullName\_SubjectCode\_JUnitTestLab.zip

Example: NguyenVanA\_CSE107\_JUnitTestLab.zip