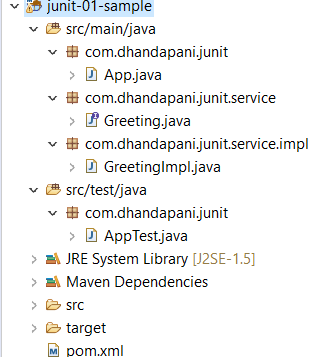
**JUNIT SAMPLES**

**Sample 1 :**

**Structure**



**App.java**

package com.dhandapani.junit;

public class App {

public static void main(String[] args) {

System.*out*.println("Hello World!");

}

}

**Greeting.java**

package com.dhandapani.junit.service;

public interface Greeting {

public String greet(String name);

}

**GreetingImpl.java**

package com.dhandapani.junit.service.impl;

import com.dhandapani.junit.service.Greeting;

public class GreetingImpl implements Greeting {

public String greet(String name) {

if (name == null || name.length() == 0) {

throw new IllegalArgumentException();

}

return "Hello " + name;

}

}

**AppTest.java**

package com.dhandapani.junit;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import com.dhandapani.junit.service.Greeting;

import com.dhandapani.junit.service.impl.GreetingImpl;

public class AppTest {

private Greeting gi;

@Before

public void setUp() {

System.*err*.println("setUp");

gi = new GreetingImpl();

}

@After

public void tearDown() {

System.*err*.println("tearDown");

gi = null;

}

@Test

public void greetShouldReturnValidOuptut() {

System.*err*.println("greetShouldReturnValidOuptut");

String result = gi.greet("JUnit");

*assertNotNull*(result);

*assertEquals*("Hello JUnit", result);

}

@Test(expected = IllegalArgumentException.class)

public void greetShouldThrowIllegalArgumentExceptionIfNameIsNull() {

System.*err*.println("greetShouldThrowIllegalArgumentExceptionIfNameIsNull");

gi.greet(null);

}

@Test(expected = IllegalArgumentException.class)

public void greetShouldThrowIllegalArgumentExceptionIfNameIsEmpty() {

System.*err*.println("greetShouldThrowIllegalArgumentExceptionIfNameIsEmpty");

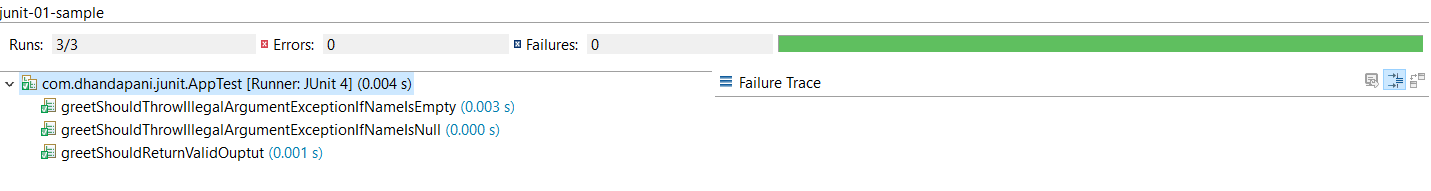
gi.greet("");

}

}

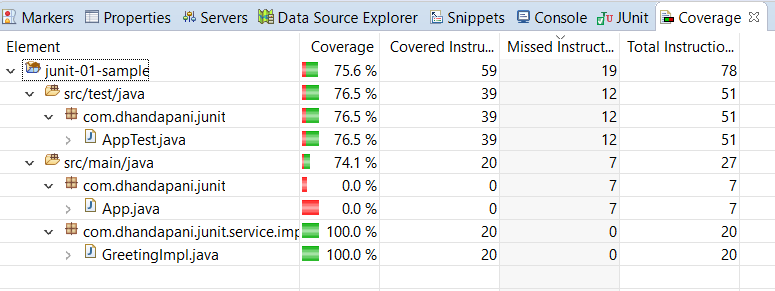
**Test Result**

Select Project -> Run As -> Junit Test



**Test/Code Coverage**

Select Project -> Coverage As -> Junit Test



**Fail Scenario**

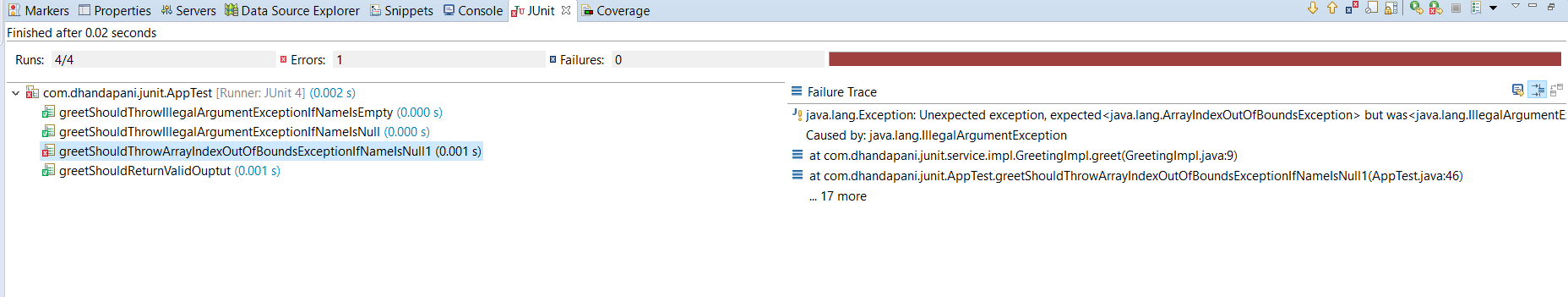
@Test(expected = ArrayIndexOutOfBoundsException.class)

public void greetShouldThrowArrayIndexOutOfBoundsExceptionIfNameIsNull1() {

System.*err*.println("greetShouldThrowArrayIndexOutOfBoundsExceptionIfNameIsNull1");

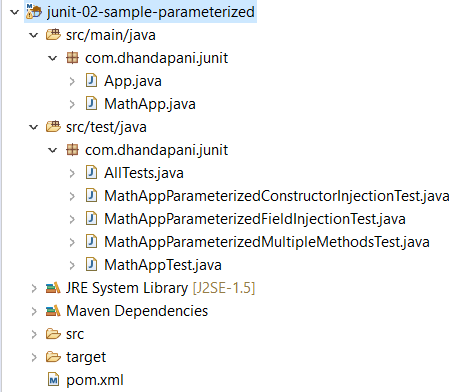
gi.greet(null);

}



**Sample 2**

**Structure**



**App.java**

package com.dhandapani.junit;

public class App {

public static void main(String[] args) {

System.*out*.println("Hello World!");

}

}

**MathApp.java**

package com.dhandapani.junit;

public class MathApp {

public int add(int a, int b) {

System.*out*.println("a = " + a + " b = " + b + " Sum of a and b = " + (a + b));

return a + b;

}

public int sub(int a, int b) {

System.*out*.println("a = " + a + " b = " + b + " Substraction of a and b = " + (a - b));

return a - b;

}

public int mul(int a, int b) {

System.*out*.println("a = " + a + " b = " + b + " Multiplication of a and b = " + (a \* b));

return a \* b;

}

}

**MathAppTest.java**

package com.dhandapani.junit;

import static org.junit.Assert.assertEquals;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class MathAppTest {

private MathApp mathApp;

// This method will be executed before the execution of each test method

@Before

public void setUp() {

mathApp = new MathApp();

}

// This method will be executed after the execution of each test method

@After

public void tearDown() {

mathApp = null;

}

@Test

public void testAddWithPositiveNumber() {

assertEquals(9, mathApp.add(4, 5));

}

@Test

public void testAddWithNegativeNumber() {

assertEquals(-1, mathApp.add(4, -5));

}

@Test

public void testSubWithPositiveNumber() {

assertEquals(1, mathApp.sub(5, 4));

}

@Test

public void testSubWithNegativeNumber() {

assertEquals(9, mathApp.sub(5, -4));

}

@Test

public void testMulWithPositiveNumber() {

assertEquals(20, mathApp.mul(5, 4));

}

@Test

public void testMulWithNegativeNumber() {

assertEquals(-20, mathApp.mul(5, -4));

}

}

**MathAppParameterizedConstructorInjectionTest.java**

package com.dhandapani.junit;

import static org.hamcrest.CoreMatchers.is;

import static org.junit.Assert.assertThat;

import java.util.Arrays;

import java.util.Collection;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

@RunWith(value = Parameterized.class)

public class MathAppParameterizedConstructorInjectionTest {

private MathApp mathApp;

private int actualInput1;

private int actualInput2;

private int expectedOutput;

public MathAppParameterizedConstructorInjectionTest(int actualInput1, int actualInput2, int expectedOutput) {

this.actualInput1 = actualInput1;

this.actualInput2 = actualInput2;

this.expectedOutput = expectedOutput;

}

// This method will be executed before the execution of each test method

@Before

public void setUp() {

mathApp = new MathApp();

}

// This method will be executed after the execution of each test method

@After

public void tearDown() {

mathApp = null;

}

/\*

\* @Parameters(name="{index}: testAdd({0}+{1})={2}") public static

\* Collection<Object[]> testData() { return Arrays.asList(new Object[][] { { 1,

\* 1, 2 }, { 5, 6, 11 } }); }

\*/

@Parameters

public static Collection<Integer[]> testData() {

Integer expectedOuputs[][] = { { 5, 5, 10 }, { 5, 4, 9 } };

return Arrays.asList(expectedOuputs);

}

@Test

public void testAddition() {

assertThat(mathApp.add(actualInput1, actualInput2), is(expectedOutput));

}

}

**MathAppParameterizedFieldInjectionTest.java**

package com.dhandapani.junit;

import static org.hamcrest.CoreMatchers.is;

import static org.junit.Assert.\*;

import java.util.Arrays;

import java.util.Collection;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

@RunWith(value = Parameterized.class)

public class MathAppParameterizedFieldInjectionTest {

private MathApp mathApp;

private int actualInput1;

private int actualInput2;

private int expectedOutput;

public MathAppParameterizedFieldInjectionTest(int actualInput1, int actualInput2, int expectedOutput) {

this.actualInput1 = actualInput1;

this.actualInput2 = actualInput2;

this.expectedOutput = expectedOutput;

}

@Before

public void setUp() {

mathApp = new MathApp();

}

@After

public void tearDown() {

mathApp = null;

}

@Parameters

public static Collection<Integer[]> testData() {

Integer expectedOuputs[][] = { { 5, 5, 10 }, { 5, 4, 9 } };

return Arrays.asList(expectedOuputs);

}

@Test

public void testAddition() {

assertThat(mathApp.add(actualInput1, actualInput2), is(expectedOutput));

}

}

**MathAppParameterizedMultipleMethodsTest.java**

package com.dhandapani.junit;

import static org.hamcrest.CoreMatchers.is;

import static org.junit.Assert.assertThat;

import java.util.Arrays;

import java.util.Collection;

import org.junit.After;

import org.junit.Assume;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

@RunWith(value = Parameterized.class)

public class MathAppParameterizedMultipleMethodsTest {

private MathApp mathApp;

enum MethodType {

SUBSTRACTION, ADDITION, MULTIPLICATION

};

private int actualInput1;

private int actualInput2;

private int expectedOutput;

private MethodType type;

public MathAppParameterizedMultipleMethodsTest(int actualInput1, int actualInput2, int expectedOutput,

MethodType type) {

this.actualInput1 = actualInput1;

this.actualInput2 = actualInput2;

this.expectedOutput = expectedOutput;

this.type = type;

}

@Before

public void setUp() {

mathApp = new MathApp();

}

@After

public void tearDown() {

mathApp = null;

}

@Parameters(name = "{index}: test( {0} {3} {1} )= {2}")

public static Collection<Object[]> testData() {

Object expectedOuputs[][] = { { 5, 5, 10, MethodType.ADDITION }, { 5, 4, 1, MethodType.SUBSTRACTION },

{ 5, 4, 20, MethodType.MULTIPLICATION }, { 5, 6, 11, MethodType.ADDITION },

{ 4, -5, 9, MethodType.SUBSTRACTION }, { -5, -5, 25, MethodType.MULTIPLICATION } };

return Arrays.asList(expectedOuputs);

}

@Test

public void testAdd() {

Assume.assumeTrue(type == MethodType.ADDITION);

assertThat(mathApp.add(actualInput1, actualInput2), is(expectedOutput));

}

@Test

public void testSub() {

Assume.assumeTrue(type == MethodType.SUBSTRACTION);

assertThat(mathApp.sub(actualInput1, actualInput2), is(expectedOutput));

}

@Test

public void testMul() {

Assume.assumeTrue(type == MethodType.MULTIPLICATION);

assertThat(mathApp.mul(actualInput1, actualInput2), is(expectedOutput));

}

}

**AllTests.java**

package com.dhandapani.junit;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

@SuiteClasses({ MathAppParameterizedConstructorInjectionTest.class, MathAppParameterizedFieldInjectionTest.class,

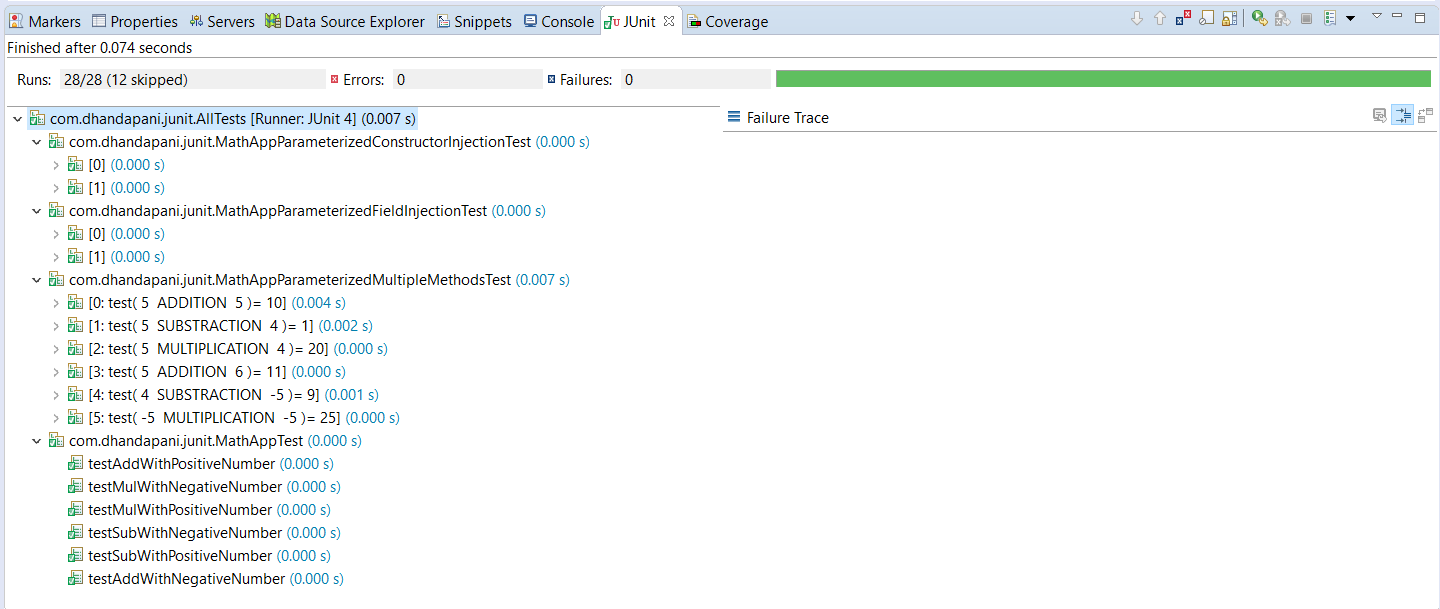
MathAppParameterizedMultipleMethodsTest.class, MathAppTest.class })

public class AllTests

{

}

**Test Result**



**Test / Code Coverage**

