**JUnit**

**@Test** - To mark a method as a test method

**@Test(expected=ExceptionClassName.class)** - To mark a method for the exception scenario

**@Before** - Will excute before each test method

**@After** - Will execute after each test method

**@BeforeClass** - Will run before entire test class

**@AfterClass** - Will run after entire test class

**@Ignore** - If we use this annontation on the method that method will be ignored

**static org.junit.Assert.\*** - To use the static assert methods

Return type of every Assert method is void

**Methods of Assert:**

1. void assertNull(Object obj) - Checks that the object is null

2. void assertNotNull(Object obj) - Checks that the object is not null

3. void assertEquals(boolean expected, boolean actual) - Checks that two primitives/objects are equal

4. void assertTrue(boolean condition) - Checks that the condition is true

5. void assertFalse(boolean condition) - Checks that the condition is false

6. void assertSame(obj1, obj2) - Checks that two object references are pointing to the same Object

7. void assertNotSame(obj1, obj2) - checks that two object references are not pointing to the same Object

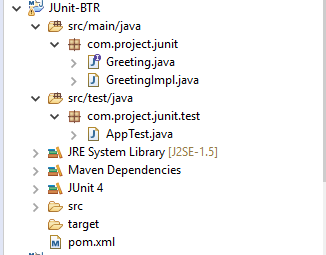
8. void assertArrayEquals(expectedArray, actualArray) - Checks that the two arrays are equal

@RunWith(SpringJUnit4ClassRunner.class)

It extends BlockJUnit4ClassRunner.class

Implements JunitRunner framework for writing Integration Test

**Example:**



**Greeting.java**

package com.project.junit;

public interface Greeting

{

public String greet(String name);

}

**GreetingImpl.java**

package com.project.junit;

public class GreetingImpl implements Greeting

{

public String greet(String name)

{

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

{

throw new IllegalArgumentException();

}

return "Hello "+name;

}

}

**App.java**

package com.project.junit.test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import com.project.junit.Greeting;

import com.project.junit.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("");

}

}

**Output:**

All unit test is passed

setUp

greetShouldThrowIllegalArgumentExceptionIfNameIsEmpty

tearDown

setUp

greetShouldThrowIllegalArgumentExceptionIfNameIsNull

tearDown

setUp

greetShouldReturnValidOuptut

tearDown

Test Coverage / Code Coverage:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TC or CC is the number of lines of our source code that is tested whenever unit tests are run against the source code.

We measure CC using tools.

Popular Tools:

1. EclEmma

2. Cobertura

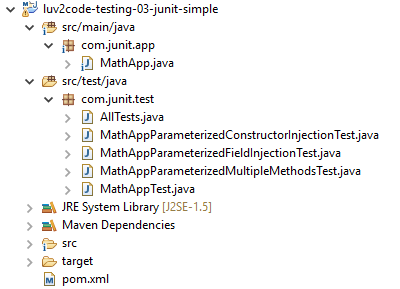
3. JTest

Parameterized Test

To use parameterized test we have to annotate the class name with @RunWith(Parameterized.class)

We can use parameterized test for testing a single method with different scenarios or inputs

Simple Example



pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>junit.mockito</groupId>

<artifactId>luv2code-testing-03-junit-simple</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>luv2code-testing-03-junit-simple</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

MathApp.java

package com.junit.app;

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.junit.test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import com.junit.app.MathApp;

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));

}

}

MathAppParameterizedConstructorInjection.java

package com.junit.test;

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;

import com.junit.app.MathApp;

@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));

}

}

MathAppParameterizedFieldInjection.java

package com.junit.test;

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.Parameter;

import org.junit.runners.Parameterized.Parameters;

import com.junit.app.MathApp;

@RunWith(value = Parameterized.class)

public class MathAppParameterizedFieldInjectionTest

{

// Field Injection the fields should be public

private MathApp mathApp;

@Parameter(value=0)

public int actualInput1;

@Parameter(value=1)

public int actualInput2;

@Parameter(value=2)

public int 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(name="{index}: testAdd({0}+{1})={2}")

public static Collection<Integer[]> testData()

{

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

return Arrays.asList(expectedOuputs);

}

@Test

public void testAdd()

{

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

}

}

MathAppParameterizedMultipleMethods.java

package com.junit.test;

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.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;

import com.junit.app.MathApp;

@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;

}

// 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}: 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 (TestSuite)

package com.junit.test;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

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

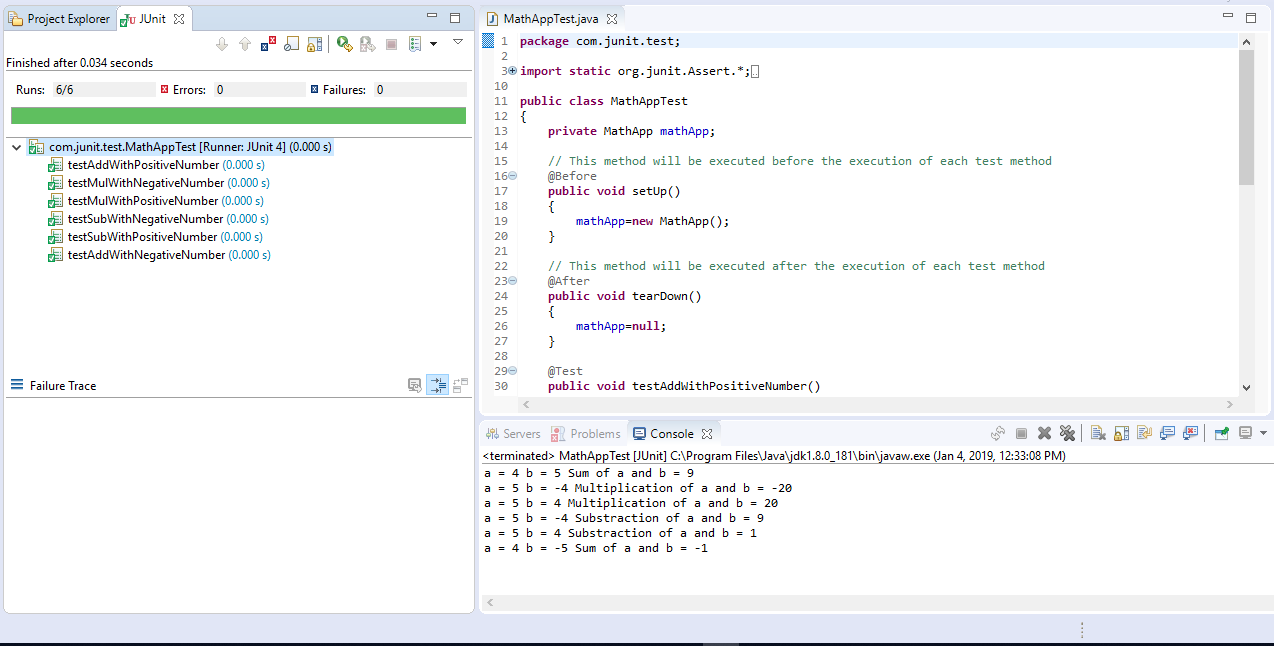
MathAppParameterizedMultipleMethodsTest.class, MathAppTest.class })

public class AllTests

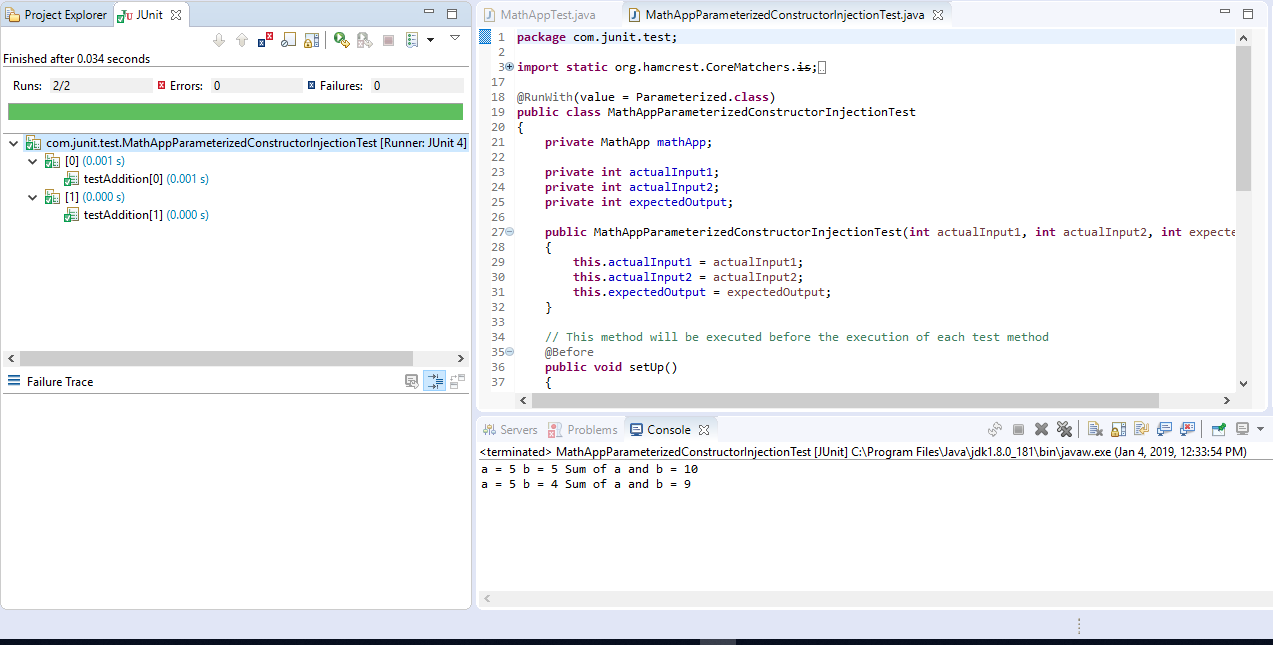
{

}

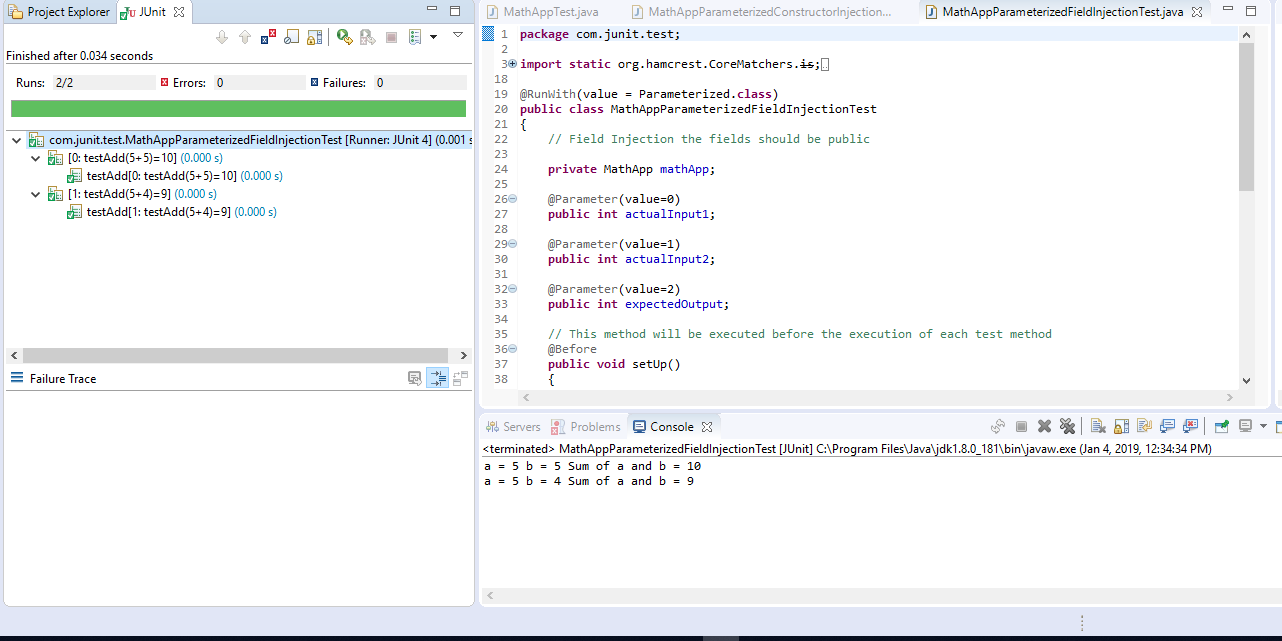
Output MathAppTest.java



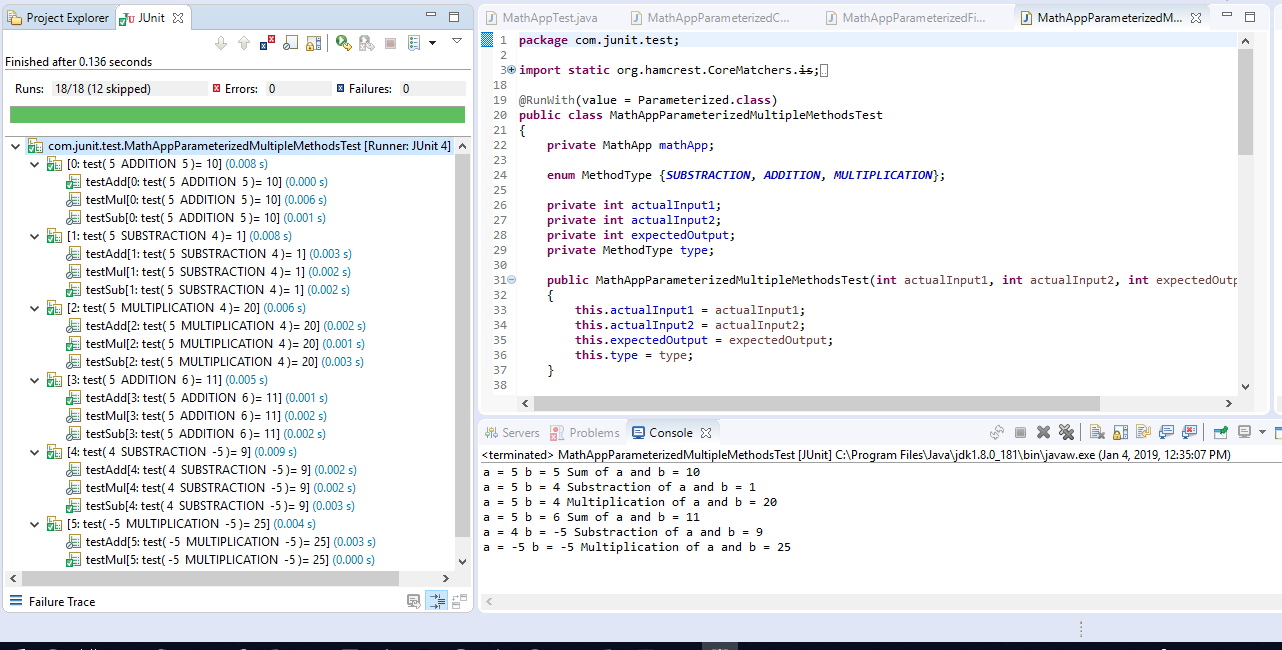
Output MathAppParameterizedConstructorInjectionTest.java



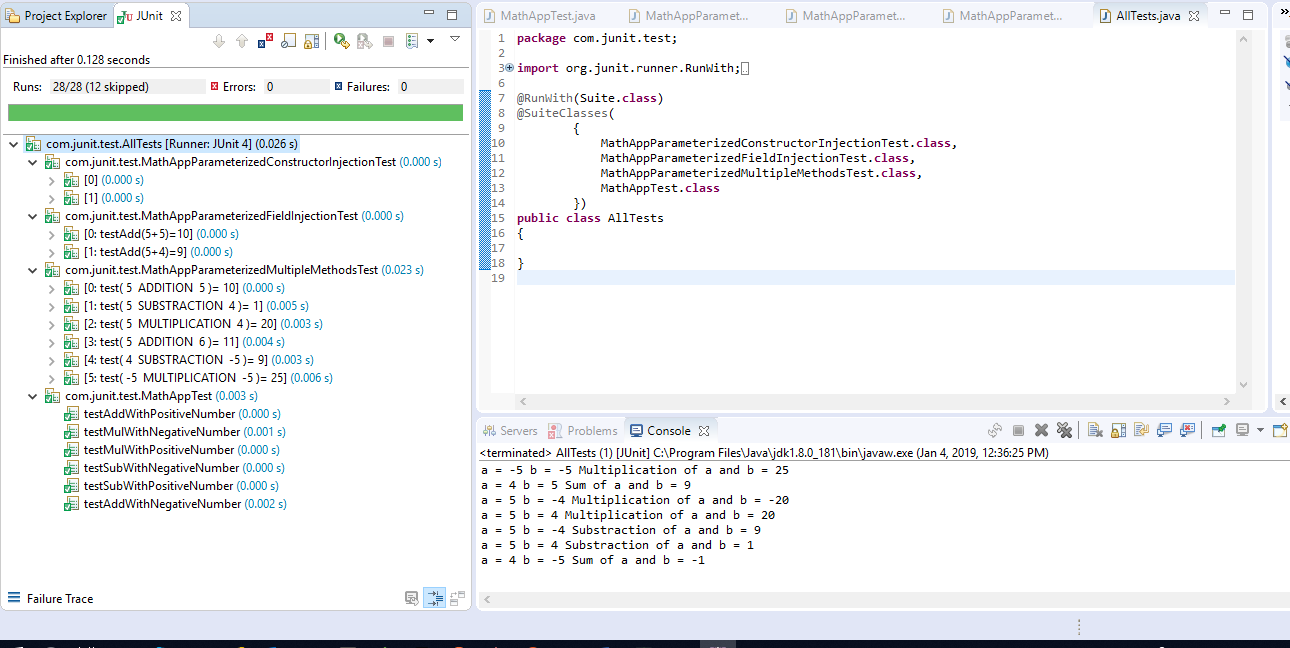
Output MathAppParameterizedFieldInjectionTest.java



Output MathAppParameterizedMultipleMethodsTest.java



Output AllTests.java



**Mockito**

**Mocking** - Testing in Isolation, Testing our code without testing our dependency(Dependencies are tested automatically)

A------------------->B Mock

**Example Scenario:**

If class A depends on or uses Class B and if we are unit testing class A then we will mockout Class B and all of its methods that class A uses.

The "**Mock**" object will takes the place of real object of Class B

ReservationController (UI Components)

|

ReservationBO (Business Logics)

|

ReservationDAO (Database Operations)

Now Testing ReservationController and mockout all the methods of ReservationBO that the controller class uses.

**Steps in Mocking**

Step 1. Stubbing

Step 2. Setting the Expectations

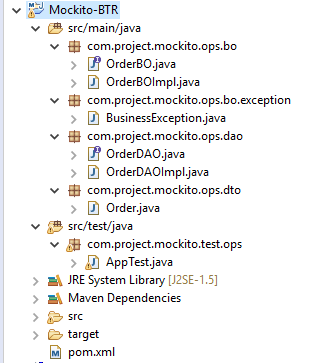
Step 3. Verifying

**Mockito Framework provides the ability to do the above 3 steps**

Step1 & Step2 or Stubbing & Setting the Expectations = @Mock (when, thenReturn and thenThrow, etc)

Verifying = verify, matchers, etc

**Example:**



**Order.java**

package com.project.mockito.ops.dto;

public class Order

{

private int id;

private String status;

public int getId()

{

return id;

}

public void setId(int id)

{

this.id = id;

}

public String getStatus()

{

return status;

}

public void setStatus(String status)

{

this.status = status;

}

@Override

public String toString()

{

return "Order [id=" + id + ", status=" + status + "]";

}

}

**OrderBO.java**

package com.project.mockito.ops.bo;

import com.project.mockito.ops.bo.exception.BusinessException;

import com.project.mockito.ops.dto.Order;

public interface OrderBO

{

boolean createOrder(Order order)throws BusinessException;

boolean cancelOrder(int id)throws BusinessException;

boolean deleteOrder(int id)throws BusinessException;

}

**OrderBOImpl.java**

package com.project.mockito.ops.bo;

import java.sql.SQLException;

import com.project.mockito.ops.bo.exception.BusinessException;

import com.project.mockito.ops.dao.OrderDAO;

import com.project.mockito.ops.dto.Order;

public class OrderBOImpl implements OrderBO

{

private OrderDAO dao;

public OrderDAO getDao()

{

return dao;

}

public void setDao(OrderDAO dao)

{

this.dao = dao;

}

public boolean createOrder(Order order) throws BusinessException

{

try

{

int create = dao.create(order);

if(create==0)

{

return false;

}

}

catch (SQLException e)

{

throw new BusinessException(e);

}

return true;

}

public boolean cancelOrder(int id) throws BusinessException

{

try

{

Order read = dao.read(id);

read.setStatus("Cancelled");

int update = dao.update(read);

if(update==0)

{

return false;

}

}

catch (SQLException e)

{

throw new BusinessException(e);

}

return true;

}

public boolean deleteOrder(int id) throws BusinessException

{

try

{

int delete = dao.delete(id);

if(delete==0)

{

return false;

}

}

catch (SQLException e)

{

throw new BusinessException(e);

}

return true;

}

}

**OrderDAO.java**

package com.project.mockito.ops.dao;

import java.sql.SQLException;

import com.project.mockito.ops.dto.Order;

public interface OrderDAO

{

public int create(Order order)throws SQLException;

public Order read(int id)throws SQLException;

public int update(Order order)throws SQLException;

public int delete(int id)throws SQLException;

}

**OrderDAOImpl.java**

package com.project.mockito.ops.dao;

import java.sql.SQLException;

import com.project.mockito.ops.dto.Order;

public class OrderDAOImpl implements OrderDAO

{

public int create(Order order) throws SQLException

{

return 0;

}

public Order read(int id) throws SQLException

{

return null;

}

public int update(Order order) throws SQLException

{

return 0;

}

public int delete(int id) throws SQLException

{

return 0;

}

}

**BusinessException.jav**

package com.project.mockito.ops.bo.exception;

import java.sql.SQLException;

public class BusinessException extends Exception

{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public BusinessException(SQLException e)

{

super(e);

}

}

**AppTest.java**

package com.project.mockito.test.ops;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.\*;

import java.sql.SQLException;

import com.project.mockito.ops.bo.OrderBOImpl;

import com.project.mockito.ops.bo.exception.BusinessException;

import com.project.mockito.ops.dao.OrderDAO;

import com.project.mockito.ops.dto.Order;

public class AppTest

{

/\*\*

\* Stubbing out the dependency

\*/

@Mock

private OrderDAO dao;

private OrderBOImpl obo;

@Before

public void setUp()

{

MockitoAnnotations.initMocks(this);

obo=new OrderBOImpl();

obo.setDao(dao);

}

@After

public void tearDown()

{

obo=null;

}

@Test

public void placeOrderShouldCrateAnOrder() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.create(order)).thenReturn(new Integer(1));

boolean createOrder = obo.createOrder(order);

assertTrue(createOrder);

/\*\*

\* Verifying the result

\*/

verify(dao).create(order);

}

@Test

public void placeOrderShouldNotCreateAnOrder() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.create(order)).thenReturn(new Integer(0));

boolean createOrder = obo.createOrder(order);

assertFalse(createOrder);

/\*\*

\* Verifying the result

\*/

verify(dao).create(order);

}

@Test(expected = BusinessException.class)

public void placeOrderShouldThrowBusinessException() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.create(order)).thenThrow(SQLException.class);

boolean createOrder = obo.createOrder(order);

}

@Test

public void cancelOrderMethodShouldCancelTheOrder() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.read(1)).thenReturn(order);

when(dao.update(order)).thenReturn(new Integer(1));

boolean cancelOrder = obo.cancelOrder(1);

assertTrue(cancelOrder);

/\*\*

\* Verifying the result

\*/

verify(dao).read(1);

verify(dao).update(order);

}

@Test

public void cancelOrderMethodShouldNotCancelTheOrder() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.read(123)).thenReturn(order);

when(dao.update(order)).thenReturn(new Integer(0));

boolean cancelOrder = obo.cancelOrder(123);

assertFalse(cancelOrder);

/\*\*

\* Verifying the result

\*/

verify(dao).read(123);

verify(dao).update(order);

}

@Test(expected = BusinessException.class)

public void cancelOrderMethodShouldThrowBusinessExceptionOnRead() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.read(123)).thenThrow(SQLException.class);

obo.cancelOrder(123);

}

@Test(expected = BusinessException.class)

public void cancelOrderMethodShouldThrowBusinessExceptionOnUpdate() throws SQLException, BusinessException

{

Order order=new Order();

/\*\*

\* Setting up the expectations

\*/

when(dao.read(123)).thenReturn(order);

when(dao.update(order)).thenThrow(SQLException.class);

boolean cancelOrder = obo.cancelOrder(123);

/\*\*

\* Verifying the result

\*/

verify(dao).update(order);

}

}

