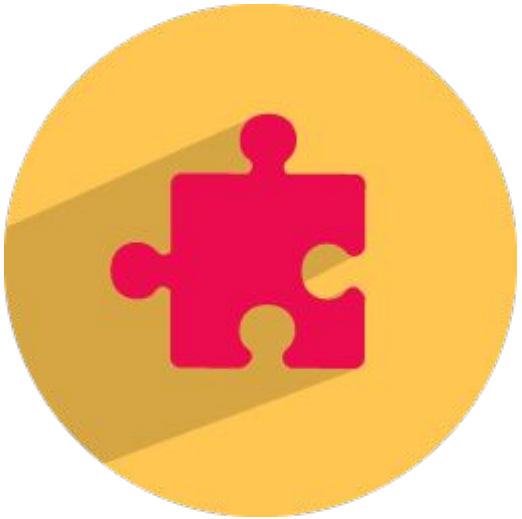
Write Unit Testing with JUnit

pokpitch.patcharadamrongkul@stream.co.th

Stream
it consulting

Introduction to Unit Testing



Unit testing is a software testing method by which individual units of source code.

“Focus on testing whether a method follows the terms of its *API contract*”

“Confirm that the method accepts the expected range of input and that the returns the expected value for each input”

Starting from scratch

The simple calculator class

```
public class Calculator {
        public double add(double number1, double number2) {

                return number1 + number2;

        }

}
```

A Simple Test Calculator Program

```
public class CalculatorTest {
        public static void main(String[] args) {
                Calculator calculator = new Calculator();
                Double result = calculator.add(10,50);
                if(result != 60) {
                        System.out.println(“Bad result: ” + result);
                }
        }
}
```

A (Slightly) Better Test Program

```
public class CalculatorTest {

    private int nbError = 0;

    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        if(result != 60) {
            throw new IllegalStateException("Bad result: " + result);
        }
    }
```

A (Slightly) Better Test Program (Cont)

```
        public static void main(String[] args) {
                CalculatorTest test = new CalculatorTest();
                try {
                        test.testAdd();
                } catch (Throwable e) {
                        test.nbError++;
                        e.printStackTrace();
                }
                if(test.nbError > 0) {
                        throw new IllegalStateException("There were " + test.nbError + "
                error(s)");
                }
        }
}
```

Unit Testing Best Practices

- Always Write Isolated Test Case
- Test One Thing Only In One Test Case
- Use a Single Assert Method per Test Case
- Use a Naming Conversions for Test Cases
- Use Arrange-Act-Assert or Given-When-Then Style

What is JUnit?

is a simple, open source framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

Latest Stable Version: 4.12 (18 April 2016)

Understanding Unit Testing frameworks

Unit testing frameworks should follow several best practices. These seemingly minor improvements in the *CalculatorTest* program highlight **three** rules that all unit testing frameworks should follow.

Each unit test run independently of all other unit tests.

The framework should detect and report errors test by test.

It should be easy to define which unit tests will run.

Run JUnit Test Case (Command line)

```
D:\experiment\stream-it-junit4-training\002-junit-calculator-test>javac -cp junit-4.12.jar *.java

D:\experiment\stream-it-junit4-training\002-junit-calculator-test>ls
Calculator.class  CalculatorTest.class  hamcrest-core-1.3.jar
Calculator.java   CalculatorTest.java   junit-4.12.jar
```

```
D:\experiment\stream-it-junit4-training\002-junit-calculator-test>java -cp .;junit-4.12.jar;hamcrest
-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest
JUnit version 4.12
.
Time: 0.007

OK (1 test)


D:\experiment\stream-it-junit4-training\002-junit-calculator-test>
```

JUnit design goals

The JUnit team has defined three discrete goals for the framework

The framework must help us write useful tests.

The framework must help us create tests
that retain their value over time.

The framework must help us lower the cost of
writing tests by reusing code.

Downloading and Installing JUnit

Plain-Old JAR

maven

MAVEN

Plain-Old JAR

Download the following JARs and put them on your test classpath

- junit.jar
- hamcrest-core.jar

MAVEN

Add a dependency to junit:junit in test scope

pom.xml

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>unit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Testing with JUnit

JUnit has many features that make it easy to write and run tests. You'll see these features at work throughout this example.

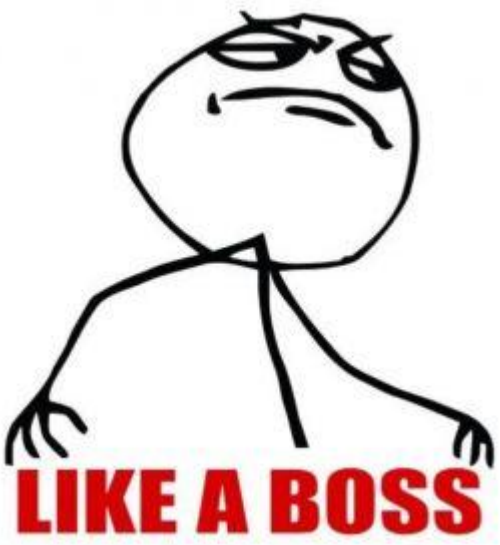
- Separate test class instances and class loaders for each unit test to avoid side effect.
- JUnit annotations to provide resource initialization and reclamation methods: @Before, @BeforeClass, @After and @AfterClass
- A variety of assert methods to make it easy to check the results of your tests.
- Integration with popular tools like Ant and Maven, and popular IDEs like Eclipse, NetBeans, IntelliJ, and JBuilder

The JUnit CalculatorTest Program

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {
        @Test
        public void testAdd() {
                Calculator calculator = new Calculator();
                double result = calculator.add(10,50);
                assertEquals(60, result, 0);
        }
}
```

Run JUnit Test Case (Maven)




```
D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building 003-junit-calculator-test-maven-project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ 003-junit-calculator-test-maven-project ---
[WARNING] Using platform encoding (MS874 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ 003-junit-calculator-test-maven-project ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding MS874, i.e. build is platform dependent!
[INFO] Compiling 1 source file to D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ 003-junit-calculator-test-maven-project ---
[WARNING] Using platform encoding (MS874 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ 003-junit-calculator-test-maven-project ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding MS874, i.e. build is platform dependent!
[INFO] Compiling 1 source file to D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ 003-junit-calculator-test-maven-project ---
[INFO] Surefire report directory: D:\experiment\stream-it-junit4-training\003-junit-calculator-test-maven-project\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.stream.CalculatorTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.082 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 7.252 s
```

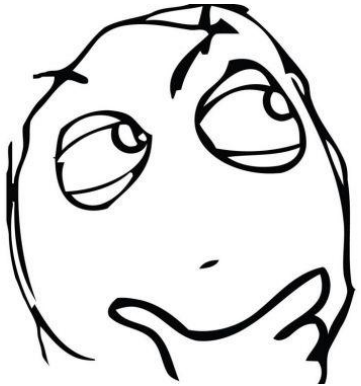
Exploring Core JUnit

We need a reliable and repeatable way to test our program.

We need to grow our test as well.

We need to make sure that we can run all of our tests at any time, no matter what code changes took place.

The Question Becomes ...



How do we run multiple test classes?

How many assert methods provided by the JUnit Assert class?

How do we find out which tests passed and which ones failed?

JUnit Assert Method Sample

assertXXX method	What it's used for
assertArrayEquals("message", A, B)	Asserts the equality of the A and B arrays.
assertEquals("message", A, B)	Asserts the equality of object A and B. This assert invokes the equals() method on the first object against the second.
assertSame("message", A, B)	Asserts that the A and B objects are the same object. (like using the == operation)
assertTrue("message", A)	Asserts that the A condition is true.
assertNotNull("message", A)	Asserts that the A object is not null.

JUnit Core Objects

JUnit Concept	Responsibilities
Assert	Lets you define the conditions that you want to test. An assert method is silent when its proposition succeeds but throws an exception if the proposition fails.
Test	A method with a @Test annotation defined a test. To run this method JUnit constructs a new instance of the containing class and then invokes the annotation method.
Test class	A test class is the container for @Test methods.
Suite	The suite allows you to group test classes together.
Runner	The Runner class runs tests. JUnit 4 is backward compatible and will run JUnit 3 tests.

Running Parameterized Tests

```
[...]
@RunWith(value=Parameterized.class)

public class ParameterizedTest {
        private double expected;
        private double valueOne;
        private double valueTwo;

        @Parameters
        public static Collection<Integer[]> getTestParameters() {
                return Arrays.asList(new Integer[][] {
```

Running Parameterized Tests (Cont)

```
                {2, 1, 1}, // expected, valueOne, valueTwo
                {3, 2, 1}, // expected, valueOne, valueTwo
                {4, 3, 1}, // expected, valueOne, valueTwo
        });
    }

    public ParameterizedTest(double expected, double
valueOne, double valueTwo) {
        this.expected  = expected;
        this.valueOne = valueOne;
        this.valueTwo = valueTwo;
    }
```

Running Parameterized Tests (Cont)

```
        @Test
        public void sum() {
                Calculator calc = new Calculator();
                assertEquals(expected, calc.add(valueOne, valueTwo), 0);
        }
}
```

Running Parameterized Tests (Cont)

```
D:\experiment\stream-it-junit4-training\004-junit-calculator-parameterized-test>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building 004-junit-calculator-parameterized-test 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ 004-junit-calculator-parameterized-test ---
[WARNING] Using platform encoding (MS874 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\experiment\stream-it-junit4-training\004-junit-calculator-parameterized-test\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ 004-junit-calculator-parameterized-test ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ 004-junit-calculator-parameterized-test ---
[WARNING] Using platform encoding (MS874 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\experiment\stream-it-junit4-training\004-junit-calculator-parameterized-test\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ 004-junit-calculator-parameterized-test ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ 004-junit-calculator-parameterized-test ---
[INFO] Surefire report directory: D:\experiment\stream-it-junit4-training\004-junit-calculator-parameterized-test\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.stream.ParameterizedTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.056 sec

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.731 s
[INFO] Finished at: 2016-05-25T00:58:11+07:00
[INFO] Final Memory: 8M/244M
[INFO] ------------------------------------------------------------------------
D:\experiment\stream-it-junit4-training\004-junit-calculator-parameterized-test>
```

JUnit Test Runners

When you're first writing tests, you want them to run as quickly and easily as possible.

You should be able to make testing part of the development cycle: code-run-test-code (or test-code run-test if you're test-first inclined)

JUnit Test Runners (Cont)

4.x version of JUnit is completely different from the 3.x versions, but JUnit 4 is built with backward compatibility.

Runner	Purpose
org.junit.internal.runners.JUnit38ClassRunner	This runner is included in the current release of JUnit only for backward compatibility. It will start the test case as a JUnit 3.8 test case.
org.junit.runners.JUnit4	This runner will start the test case as a JUnit 4 test case.
org.junit.runners.Parameterized	A Parameterized test runner runs same sets of tests with different parameters.
org.junit.runners.Suite	The Suite is a container that can hold different tests. The Suite is also a runner that executes all the @Test annotated methods in a test class.

JUnit Test Runners (Cont)

JUnit will use a default test runner of none is provided based on the test class. If you want to use specify test runner, specify the test runner class using the @RunWith annotation, as demonstrated in the following code.

```
@RunWith(value=org.junit.internal.runners.JUnit38ClassRunner.class)
public class TestWithJUnit38 extends junit.framework.TestCase {
        [...]
}
```

Composing Tests With a Suite

JUnit designed the Suite to run one or more test class. The Suite is a container used to gather tests for the purpose of grouping and invocation.

Composing Tests With a Suite

Composing a Suite from test classes

```
[...]
@RunWith(value=org.junit.internal.runners.Suite.class)
@SuiteClasses(value={FolderConfigurationTest.class,  FileConfigurationTest.class})
public class FileSystemConfigurationTestSuite {
}
```

We specify the appropriate runner with the @RunWith annotation and list the tests we want to include in this test by specifying the test classes in the @SuiteClasses annotation. All the @Test methods from these classes will be included in the Suite.

Composing a Suite of Suite

It’s possible to create a suite of test suites.

```
[...]
public class TestCaseA {
        @Test
        public void testA1() {
        }
}

[...]
public class TestCaseB {
        @Test
        public void testB1() {
        }
}
```

```
[...]
@RunWith(value=Suite.class)
@SuteClasses(value={TestCaseA.class})
public class TestSuiteA {
}

[...]
@RunWith(value=Suite.class)
@SuteClasses(value={TestCaseB.class})
public class TestSuiteB {
}

[...]
@RunWith(value=Suite.class)
@SuteClasses(value={TestSuiteA.class, TestSuiteB.class})
public class MasterTestSuite {
}
```

Software Testing Principles

The need for unit tests

The main goal of unit testing is to verify that your application works as expected and to catch bugs early.

Allow greater test coverage

Detect regressions and limiting debugging

Increasing team productivity

Refactoring with confidence

Software Testing Principles (Const)

The need for unit tests (Cont)

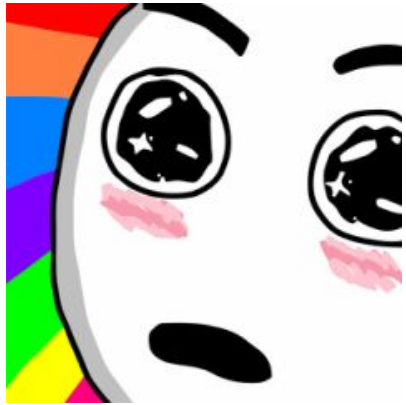
Improving Implementation

Documenting expected behaviour

Enabling code coverage and other metrics

Unit Tests as Automatic Documentation

```
import org.junit.Test;
public class TestAccount {
        [...]
        @Test(expected=AccountInsufficientFundsException.class)
        public void transferWithoutEnoughFunds() {
                long balance = 1000;
                long amountToTransfer = 2000;
                Account credit = new Account(balance);
                Account debit= new Account();
                credit.transfer(debit, amountToTransfer);
        }
}
```



The Four Type of Software Tests

Test Coverage and Development

Testing with Mock Objects

Running Unit Tests from MAVEN/ANT

Continuous Integration Tools

JUnit Extensions

Workshop