JUnit Test Case

Summary

This guide provides how to create Unit Test Case and test it using JUnit.

Description

Following table lists basics to Unit Test Case.
Detailed usage and sample project will be described shortly.
Additionally, refer to Mock Support , DB Support for Mocking or DAO Test.

Types	Description	Notes
C onstructor Test	Omit simple logic tests. However, initialization or loading must be tested.	
Setter/Getter Test	Simple setter/getter can be skipped. If there's special logic, separate the logic into another method, then test the method..	
Boundary Test	Check Null and Default values, if boundary exists, MIN/MAX check should be tested Final goal is avoiding NullPointer Exception by adding defensive logic or validity logic,	
Equality Test	Comparison between value, reference, Number, List/Array, Value Object	Effective comparison can be achieved when using JUnit with Unitils
Test Sequence handling	Method or Class Unit Test Sequence Handling	
Parameterized Test	Repeated test with parameter values changed	
Timeout Test	Set the timeout, when timeout is over, the test is failed. .	
Test Ignore	Test on Method or Classes can be skipped	

Environmental settings

For Maven Project, specify following dependencies.

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.unitils</groupId>
            <artifactId>unitils</artifactId>
            <version>2.2</version>
            <scope>test</scope>
        </dependency>
```

✔ Note: Springframework 2.5 doesn't support JUnit 4.5, use JUnit 4.4 instead.

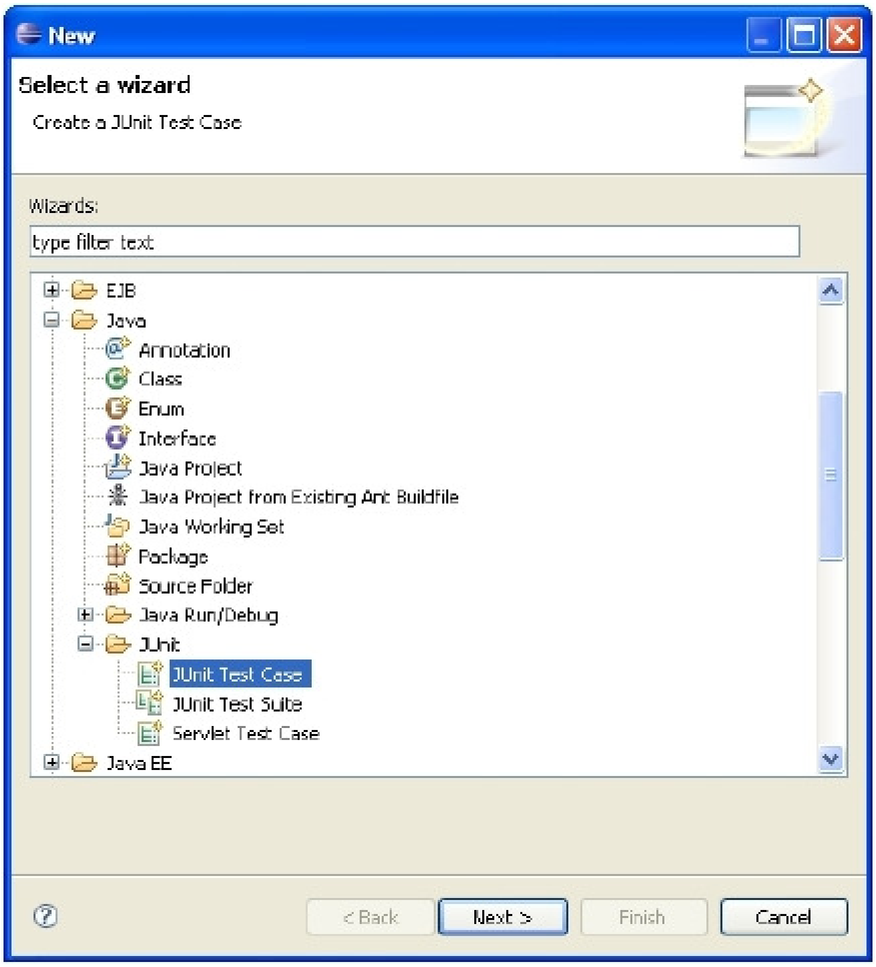
Manual

Create Test Case

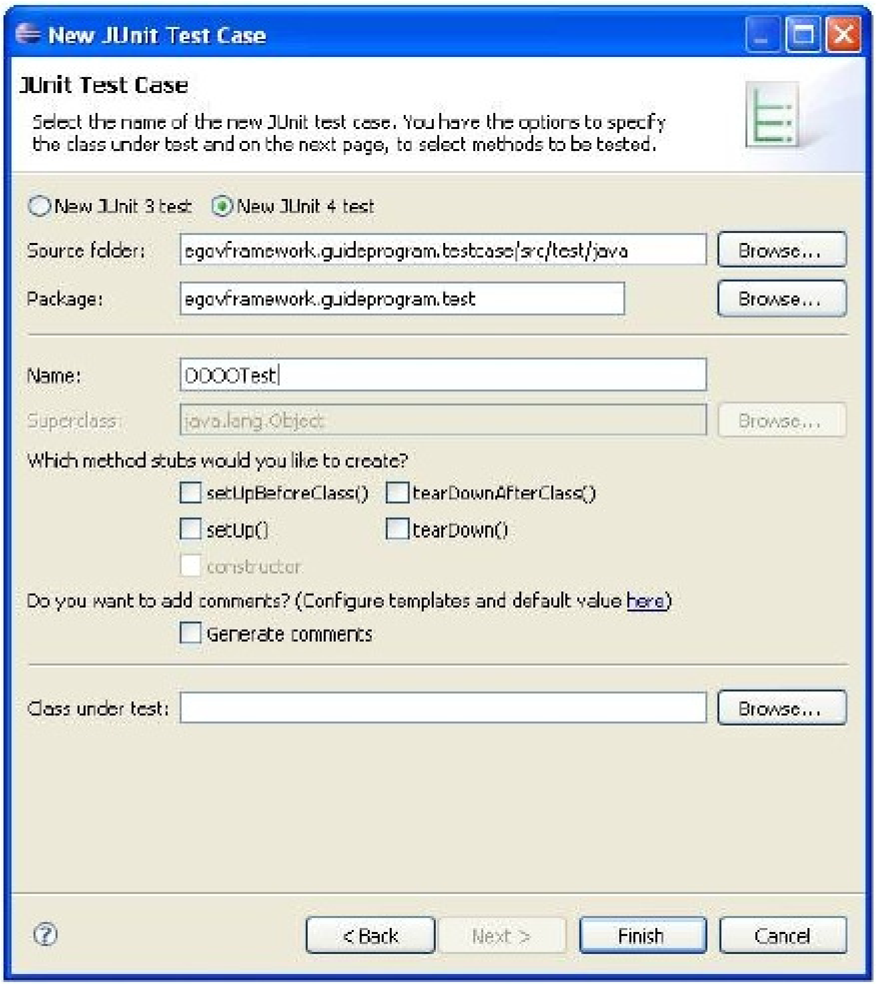
Creating Java Class by New is available, if JUnit library is used, it is automatically recognized as JUnit Test Case.

Following describes creating Test Case using Eclipse New Wizard.

1. Select JUnit Test Case, then click Next.



2. Enter Test Case name, then click Finish.
   It is widely accepted convention to add Test as suffix of target program name for Test Case. If multiple Test cases exist for target project, underscore plus sub naming is used such as OOOOTest_xxxx.
   This naming convention sets general guideline for Test case names, though it isn't mandatory.



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

New JUnit 3 test [✓ New JUnit 4 test]

Source folder: egovframework.guideprogram.testcase/src/test/java

Package: egovframework.guideprogram.test

Name: OOOOTest

Superclass: java.lang.Object

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Class under test:

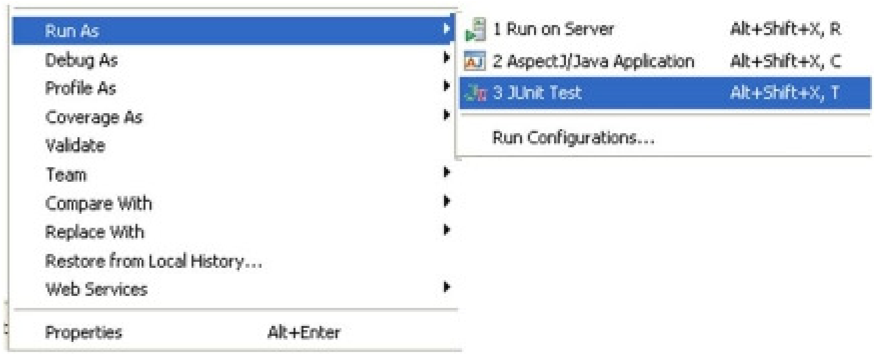
3. If you click the Create Stub method, something similar to following code will be created. (It shows how to handle before/after of all method run of test class or before/after of each method in the class)

```
public class OOOOTest {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

}
```

See samples for detailed information.

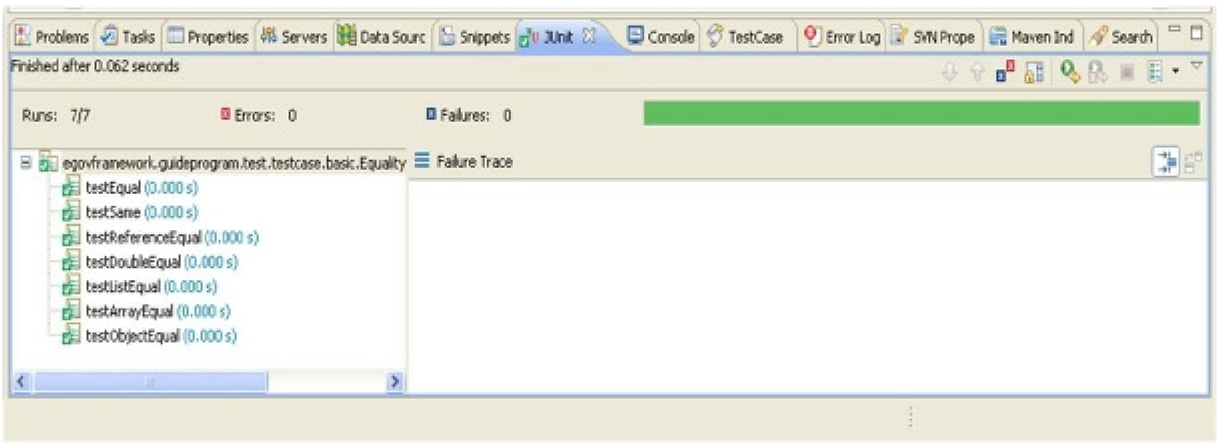
Running Test Case

Right click on test program, then select Run As… to run.


Run As
Debug As
Profile As
Coverage As
Validate
Team
Compare With
Replace With
Restore from Local History...
Web Services
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 AspectJ/Java Application Alt+Shift+X, C
3 JUnit Test Alt+Shift+X, T
Run Configurations...

Checking Test Case Result

You can review the test result as shown below. Green bar indicates successful test, otherwise red bar appears with failure reasons.



```
Finished after 0.062 seconds

Runs: 7/7        Errors: 0        Failures: 0

egovframework.guideprogram.test.testcase.basic.Equality
    testEqual (0.000 s)
    testSame (0.000 s)
    testReferenceEqual (0.000 s)
    testDoubleEqual (0.000 s)
    testListEqual (0.000 s)
    testArrayEqual (0.000 s)
    testObjectEqual (0.000 s)
```

Samples

Following programs are basic guides for developing unit test using JUnit 4. Moreover, this guide includes how to use Unitils that provides useful utilities to be used with JUnit 4.

Constructor Test

Simple logic test can be skipped. However, test for initializing values or loading should not be skipped.

Setter / Getter Test

Simple Setter/Getter test can be skipped. If special logic test is required, it is recommended to separate the logic part into another method, then test it.

Boundary Test

Goals for this test are checking Null, default value, and MIN/MAX value when ranges are provided. Desired solution is adding defensive logic or validation logic to avoid NullPointer Exceptions
.

Exception Test

Check if desired (or expected) exception actually happens. Simple annotation setup is enough to test exceptions without using full try/catch statements.

```
@Test (expected = IndexOutOfBoundsException.class)
public void testException() {
        throw new IndexOutOfBoundsException();
}
```

Equality Test

Value and reference comparison.

```
@Test
public void testReferenceEqual() {
      String string1 = new String("somevalue");
```

```
        String string2 = string1;
        assertEquals("Value of two objects are the same", string1, string2);
        assertSame("Memory address of two objects are the same ", string1, string2);
}
```

Number value comparison (basic comparison is the same, but comparing sub decimal points may vary)

```
@Test
public void testDoubleEqual() {
        double value1 = 10 / 3;
        double value2 = 3.33;
        assertEquals("equals to two decimal points", value1, value2, 2);
}
```

List/Array comparison

Default feature in JUnit returns true only when the order is the same.

```
@Test
public void testListEqual() {
        List<Integer> value1 = Arrays.asList(3, 2, 1);
        List<Integer> value2 = Arrays.asList(3, 2, 1);
        assertEquals("Sequence is the same", value1, value2);
}

@Test
public void testArrayEqual() {
        String[] value1 = new String[] {"A", "B", "C"};
        String[] value2 = new String[] {"A", "B", "C"};
        assertArrayEquals("Value and Sequence are the same", value1, value2);
}
```

✔ Using Unitils allow comparison by ignoring sequence or default values.

```
@Test
public void testReflectionEqualLenientOrder() {
        List<Integer> myList = Arrays.asList(3, 2, 1);

        assertReflectionEquals(Arrays.asList(3, 2, 1), myList);

        assertReflectionEquals("Ignore sequence", Arrays.asList(1, 2, 3), myList, LENIENT_ORDER);
}

@Test
public void testAssertReflectionEqualsIgnoringDefault() {
        assertReflectionEquals("Ignore default values”,

                                              new UserVo(1, "name", null), new UserVo(1, "name", "description1"), IGNORE_DEFAULTS);
}
```

Value Object Comparison

Value Object Comparison in JUnit requires every attribute should be compared individually.

```
 @Test
 public void testObjectEqual() {
        UserVo user1 = new UserVo(100, "name", "description");
        UserVo user2 = new UserVo(100, "name", "description");

        assertEquals(user1.getId(), user2.getId());
        assertEquals(user1.getName(), user2.getName());
        assertEquals(user1.getDescription(), user2.getDescription());
 }
```

✔ Unitils supports automatic comparison of attributes.

```
@Test
public void testAssertReflectionEqualsFieldByField() {

        UserVo user1 = new UserVo(100, "name", "description");
        UserVo user2 = new UserVo(100, "name", "description");

        assertReflectionEquals("Automatic comparison of attributes are supported ", user1, user2);
}
```

Handling before and after performing test

Handles before or after performing tests in method or class.
JUnit3.8 supported setUp(), tearDown(), however it provides following annotations to ease the control.

@BeforeClass	Performs once before any method in unit test begins.
@AfterClass	Performs once after all methods in unit test finished running.

@Before	Performs every time any method in unit test begins.
@After	Performs every time any method in unit test finishes running.

The following example declares StringBuffer and append specific string to it every time it runs to show the order of all process.

```
public class SetupTest {

        static StringBuffer sb;

        @BeforeClass
        public static void doBeforeClass() {
                sb = new StringBuffer();
                sb.append("BeforeClass/");

                assertEquals("BeforeClass/", sb.toString());
        }

        @Before
        public void doBeforeMethod() {
                sb.append("BeforeMethod/");
        }

        @After
        public void doAfterMethod() {
                sb.append("AfterMethod/");
        }

        @AfterClass
        public static void doAfterClass() {
                sb.append("AfterClass/");
                assertEquals("BeforeClass/BeforeMethod/testFirst/AfterMethod/BeforeMethod/testSecond/AfterMethod/AfterClass/", sb.toString());
        }

        @Test
        public void testFirst() {
                assertEquals("BeforeClass/BeforeMethod/", sb.toString());
                sb.append("testFirst/");
                assertEquals("BeforeClass/BeforeMethod/testFirst/", sb.toString());
        }

        @Test
        public void testSecond() {
                assertEquals("BeforeClass/BeforeMethod/testFirst/AfterMethod/BeforeMethod/", sb.toString());
                sb.append("testSecond/");
                assertEquals("BeforeClass/BeforeMethod/testFirst/AfterMethod/BeforeMethod/testSecond/", sb.toString());
        }
}
```

Parameterized Test

It is useful to test repeatedly with various parameters. Following example tests password validation, receives various password inputs and tests it all at once.

```
@RunWith(Parameterized.class)
public class ParameterizedTest {

        private String password;
        private boolean isValid;
        private static PasswordValidator validator;

        @BeforeClass
        public static void setUp() {
                validator = new PasswordValidator();
        }

        public ParameterizedTest(String password, boolean isValid) {
                this.password = password;
                this.isValid = isValid;
        }

        @Parameters
        public static Collection passwords() {
                return Arrays.asList(new Object[][] { { "1234qwer", true }, {"12345678", false}, {"1q2w3e4r", true} });
        }

        @Test
        public void isValidPasswordWithParams() {
                assertEquals(validator.isValid(this.password), this.isValid);
        }
}
```

Test target - Inner Class

```
class PasswordValidator {

        public boolean isValid(String password) {
                boolean result = false;
        int letterCnt = 0;
        int digitCnt = 0;

        for (int i = 0; i < password.length(); i++) {
            char c = password.charAt(i);
            if (Character.isLetter(c)) letterCnt++;
            else if (Character.isDigit(c)) digitCnt++;
        }

        // length should be larger than 8, one or more letter and digit character should exist.
        if (password.length() >= 8 && letterCnt > 0 && digitCnt > 0)
            result = true;

          return result;
        }
}
```

Timeout Test

Set timeout. If timeout is over, you can handle failed result.

```
@Test (timeout = 1)
public void testTimeout() {
}
```

Test Ignore

Used when you want to skip test. Class/method skips are both available.

SKIP whole test class

```
@Ignore ("Skip test (Actually, displays detailed reason)")
public class OOOOTest {
    @Test
    public void testIgnored() {
        assertTrue("All test methods are skipped", true);
    }
}
```

Skip specific method in Test Class.

```
@Ignore ("Skip test (Actually, displays detailed reason)")
@Test
public void testIgnore() {
    assertTrue("This test method is skipped", true);
}
```

References

http://www.junit.org/ [http://www.junit.org/]