JUnit - Tutorial

Lars Vogel

Version 1.2

05.04.2010

Unit testing with JUnit

This article explains unit testing with JUnit 4.x.

Table of Contents

1. Introduction

1.1. Unit Testing

A unit test is a piece of code written by a developer that tests a specific functionality in the code which is tested. Unit tests can ensure that functionality is working and can be used to validate that this functionality still works after code changes.

Unit testing uses also mocking of objects. To learn more about mock frameworks please see EasyMock Tutorial

1.2. Unit Testing with JUnit

JUnit 4.x is a test framework which uses annotation to identify the test methods. JUnit assumes is that the all test can be performed in an arbitrary order. Therefore tests should not depend other tests. To write a test with JUnit

- Annotate a method with @org.JUnit.Test
- Use a method provides by JUnit to check the expected result of the code execution versus the actual result

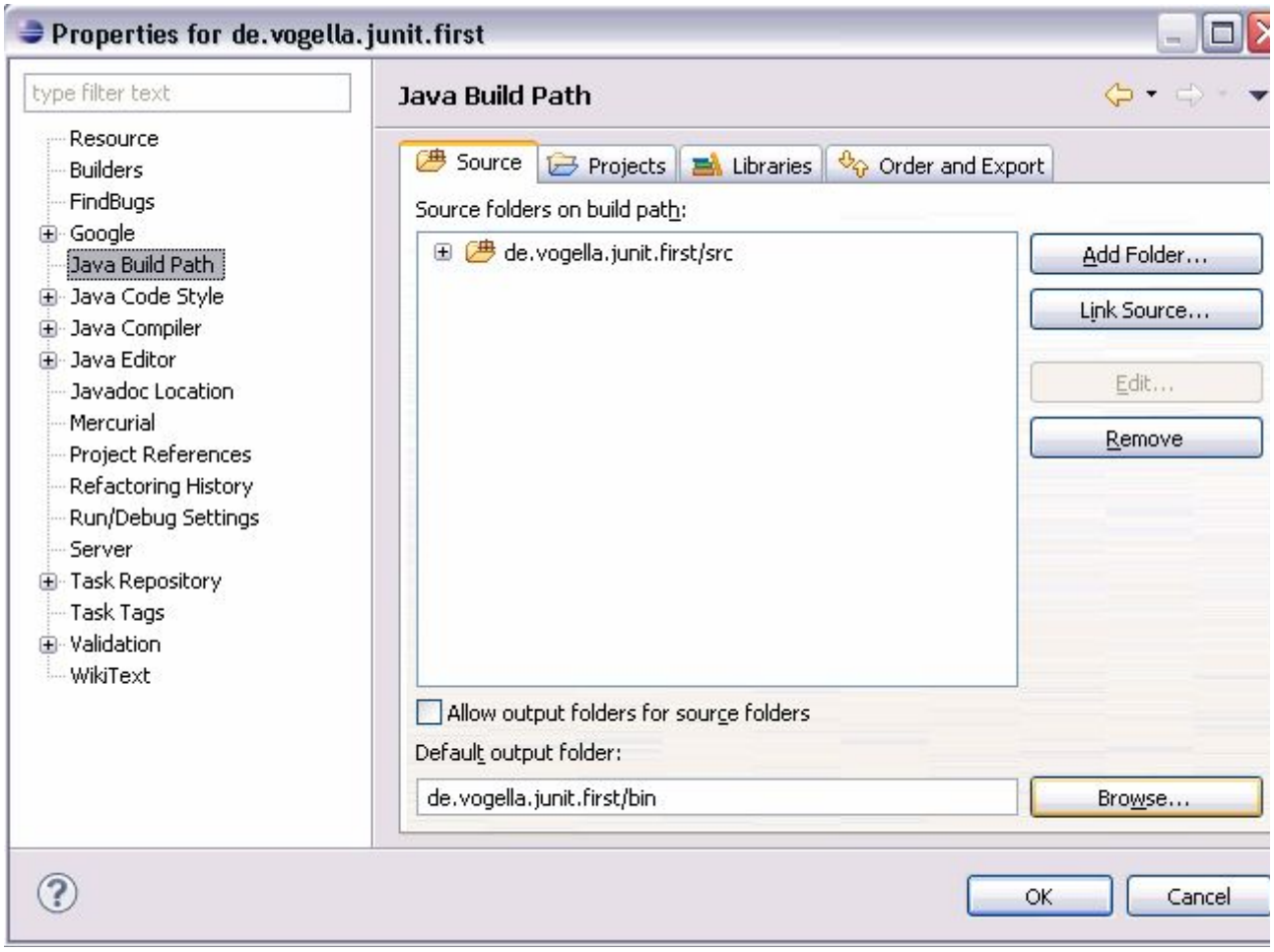
1.3. Installation of JUnit

Download JUnit4.x.jar from the JUnit website . The download contains a "junit-4.*.jar" which is the JUnit library. To make JUnit available in your Java project you have to add the the JUnit library file to your Java classpath. See Eclipse IDE Tutorial to learn how to do this in Eclipse.

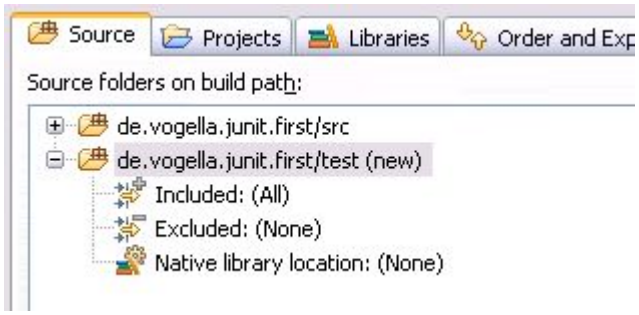
2. JUnit with Eclipse

2.1. Preparation

Create a new project "de.vogella.junit.first". We want to create the unit tests in a separate folder. Create therefore a new source folder "test" via right mouse click on your project, select properties and choose the "Java Build Path". Select the tab source code.



Press "Add folder" then then press "Create new folder". Create the folder "test".



The creation of an separate folder for the test is not mandatory. But it is good advice to keep the test coding separate from the normal coding.

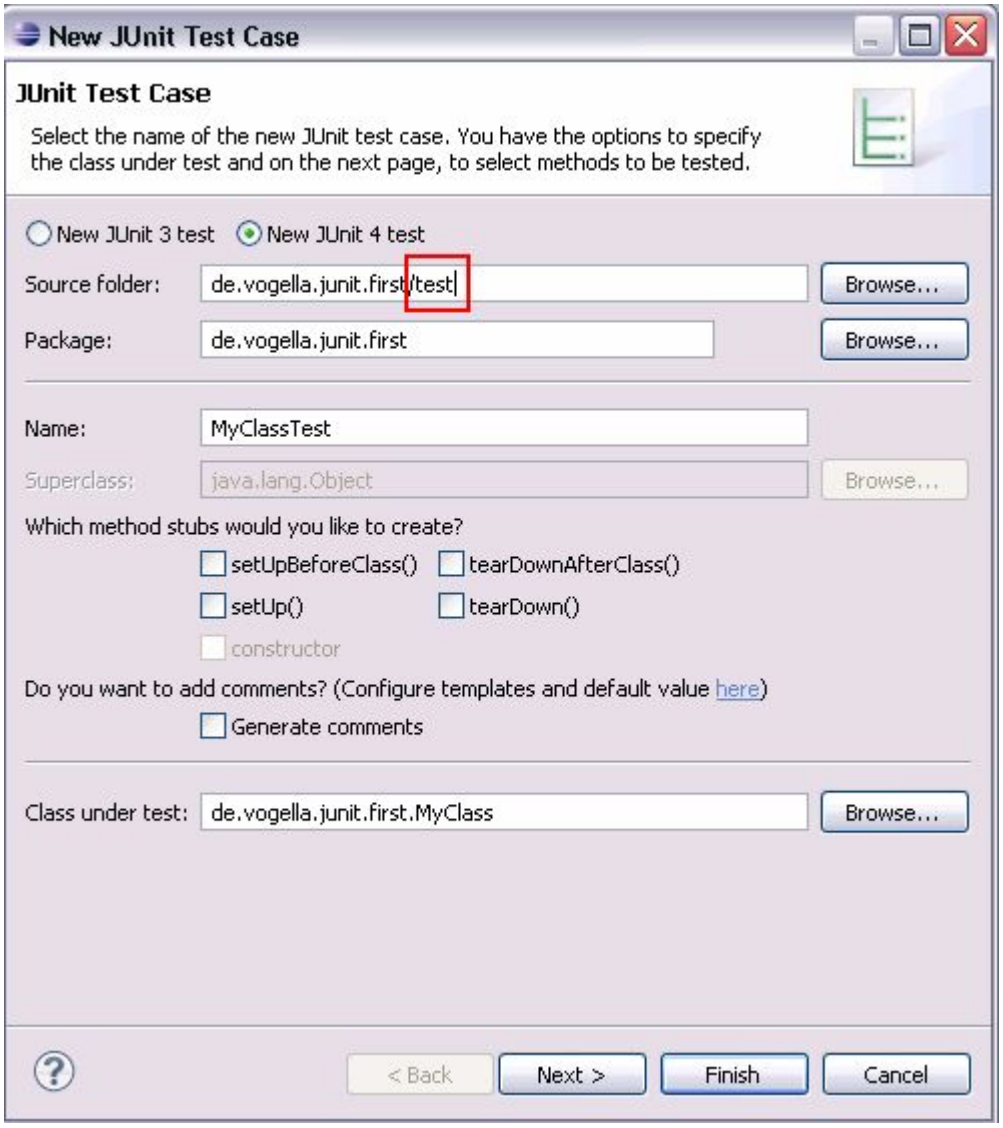
2.2. Create a Java class

Create a package "de.vogella.junit.first" and the following class.

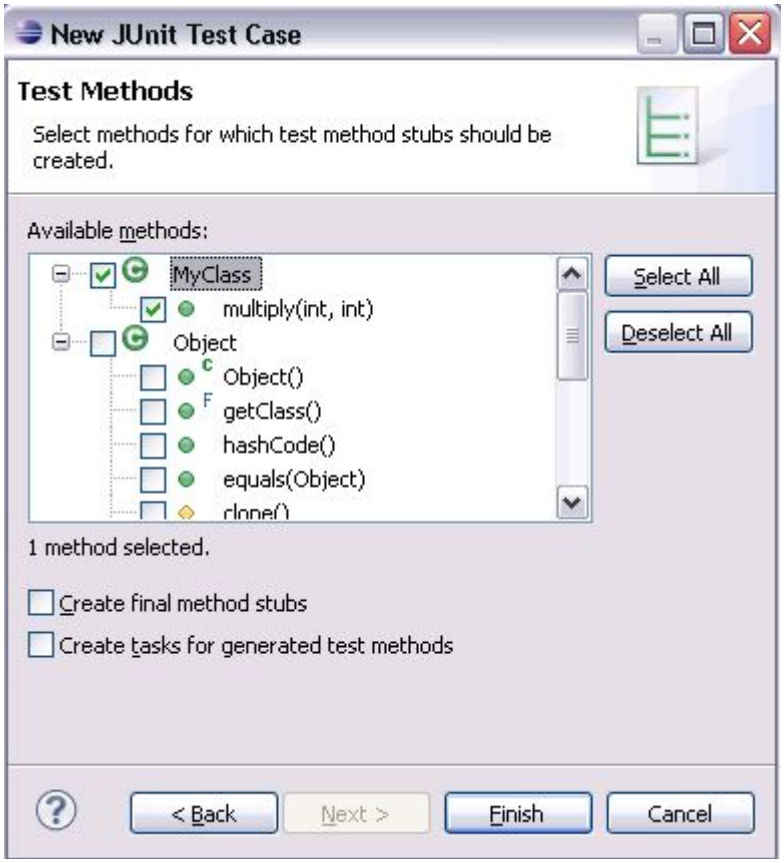
```
package de.vogella.junit.first;

public class MyClass {
        public int multiply(int x, int y) {
                return x / y;
        }
}
```

2.3. Create a JUnit test

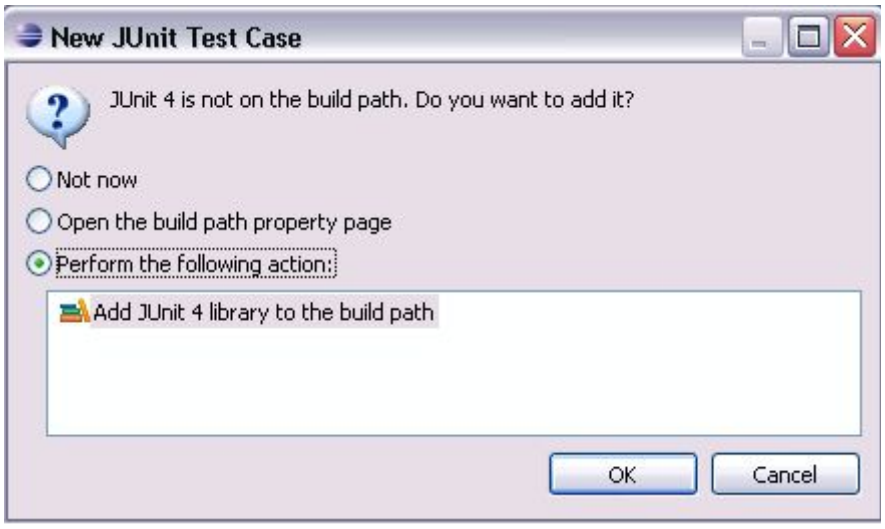
Select your new class, right mouse click and select New ->JUnit Test case, change the source folder to JUnit. Select "New JUnit 4 test". Make sure you change the source folder to test.



Press next and select the methods which you want to test.



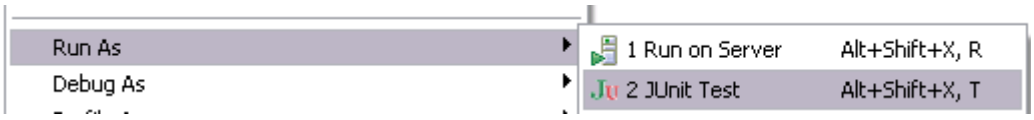
If you have not yet JUnit in your classpath, Eclipse will asked you if it should be added to the classpath.

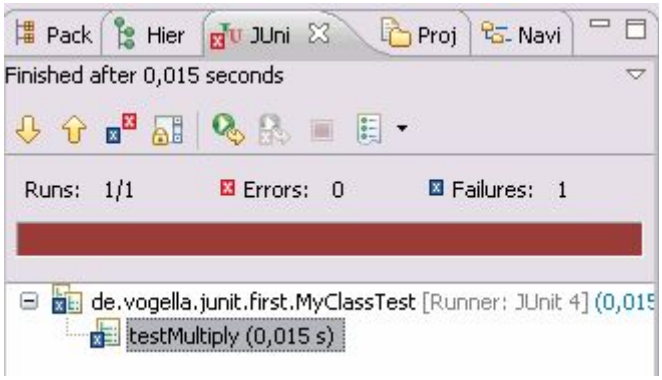


Create a test with the following code.

```
package de.vogella.junit.first;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MyClassTest {

        @Test
        public void testMultiply() {
                MyClass tester = new MyClass();
                assertEquals("Result", 50, tester.multiply(10, 5));
        }
}
```

Right click on your new test class and select Run-As-> Junit Test.

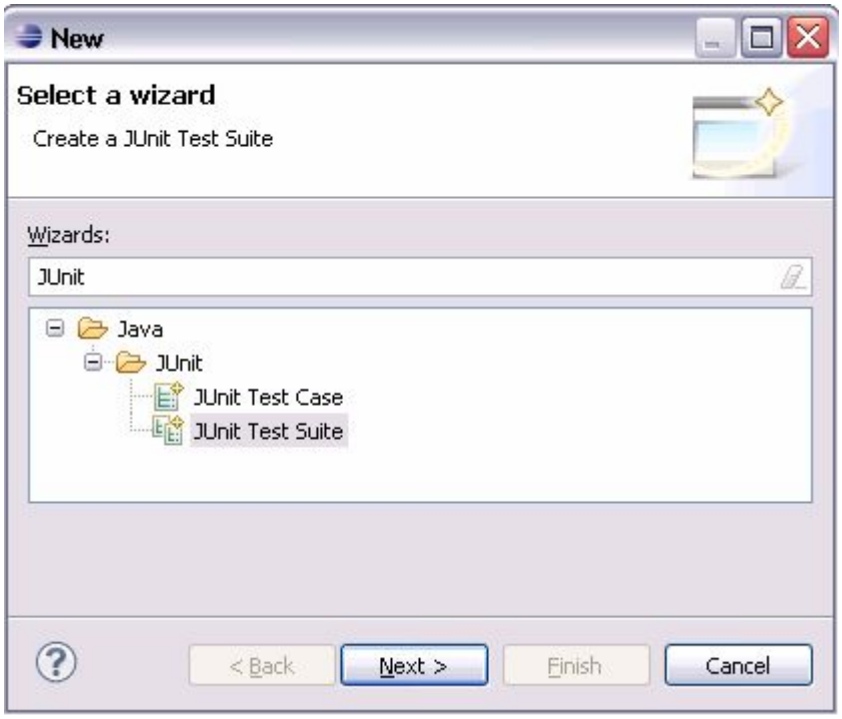




The test should be failing (indicated via a red bar). This is due to the fact that our multiplier class is currently not working correctly (it does a division instead of multiplication). Fix the bug and re-run test to get a green light.

2.4. Create a test suite

If you have several tests you can combine them into a test suite. All test in this test suite will then be executed if you run the test suite. To create a new test suite, select your test classes, right mouse click-> New-> Other -> JUnit -Test Suite



Select next and select the methods you would like to have test created for.

This does currently not work for JUnit4.0 testcases. See Bug Report

Change the coding to the following to make your test suite run your test. If you later develop another test you can add it to @Suite.SuiteClasses

```
package mypackage;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses( { MyClassTest.class })
public class AllTests {
}
```

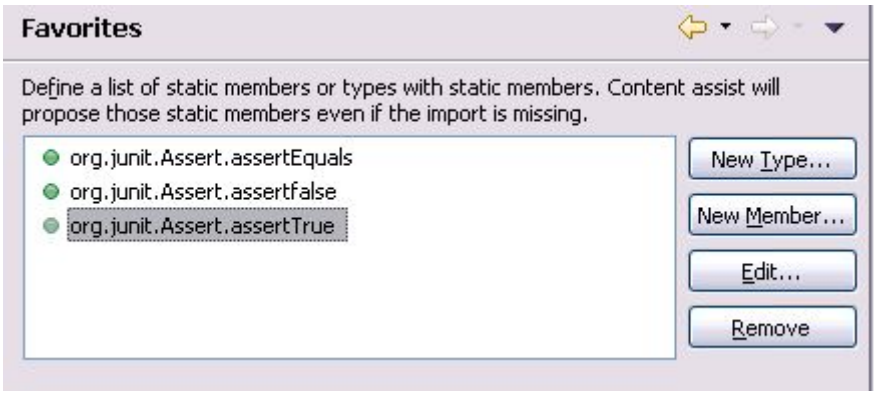
3. JUnit (more) in Detail

3.1. Static imports with Eclipse

JUnit uses a lot of static methods and Eclipse cannot automatically import static imports. You can make the JUnit test methods available via the content assists.

Open the Preferences via Window -> Preferences and select Java > Editor > Content Assist > Favorites. Add then via "New Member" the methods you need. For example this

makes the assertTrue, assertFalse and assertEquals method available.



You can now use Content Assist (Ctrl+Space) to add the method and the import.

I suggest to add at least the following new members.

- org.junit.Assert.assertTrue
- org.junit.Assert.assertFalse
- org.junit.Assert.assertEquals
- org.junit.Assert.fail

3.2. Annotations

The following give an overview of the available annotations in JUnit 4.x

Table 1. Annotations

Annotation	Description
@Test public void method()	Annotation @Test identifies that this method is a test method.
@Before public void method()	Will perform the method() before each test. This method can prepare the test environment, e.g. read input data, initialize the class)
@After public void method()	Test method must start with test
@BeforeClass public void method()	Will perform the method before the start of all tests. This can be used to perform time intensive activities for example be used to connect to a database
@AfterClass public void method()	Will perform the method after all tests have finished. This can be used to perform clean-up activities for example be used to disconnect to a database

Annotation	Description
@Ignore	Will ignore the test method, e.g. useful if the underlying code has been changed and the test has not yet been adapted or if the runtime of this test is just to long to be included.
@Test(expected=IllegalArgumentException.class)	Tests if the method throws the named exception
@Test(timeout=100)	Fails if the method takes longer then 100 milliseconds

3.3. Assert statements

The following gives an overview of the available test methods:

Table 2. Test methods

Statement	Description
fail(String)	Let the method fail, might be usable to check that a certain part of the code is not reached.
assertTrue(true);	True
assertsEquals([String message], expected, actual)	Test if the values are the same. Note: for arrays the reference is checked not the content of the arrays
assertsEquals([String message], expected, actual, tolerance)	Usage for float and double; the tolerance are the number of decimals which must be the same
assertNull([message], object)	Checks if the object is null
assertNotNull([message], object)	Check if the object is not null
assertSame([String], expected, actual)	Check if both variables refer to the same object
assertNotSame([String], expected, actual)	Check that both variables refer not to the same object
assertTrue([message], boolean condition)	Check if the boolean condition is true.

4. Thank you

Thank you for practicing with this tutorial.

I maintain this tutorial in my private time. If you like the information please help me by donating or by recommending this tutorial to other people.

5. Questions and Discussion

Before posting questions, please see the vogella FAQ . If you have questions or find an error in this article please use the www.vogella.de Google Group . I have created a short list how to create good questions which might also help you. .

6. Links and Literature

6.1. JUnit Resources

http://www.junit.org/ JUnit Homepage

6.2. vogella Resources

Eclipse Tutorials

Web development Tutorials

Android Development Tutorial

GWT Tutorial

Eclipse RCP Tutorial