Tutorials for Hibernate, EJB 2, EJB 3
Struts, JavaServerfaces (JSF)
Tomcat, JBoss, Myeclipse, Eclipse and other

Navigation

Search

Homepage
Blog
Tutorials
Schedule
Hibernate, EJB 2, JDBC Tutorials
EJB 3 development
Spring framework
General, Java technologies
Web frameworks, JSP and Servlets Tutorials
Struts 1.x
JavaServer Faces Tutorials
Debugging, Testing, Tuning
Application Server administration and configuration
Tutoriales (español)
Bugs and Exceptions
References
Hibernate 3 / JPA book and ebook
Tutorial and EBook Shop
Training - Development - Support
Links/Tips
Feedback
Disclaimer
Social projects

Eclipse Junit testing tutorial >

JUnit is a simple Java testing framework to write tests for you Java application. This tutorial gives you an overview of the features of JUnit and shows a little example how you can write tests for your Java application.

General

Author:

Sascha Wolski

Sebastian Hennebrueder

http://www.lalilluna.de/tutorials.html ? Tutorials for Struts, EJB, xdoclet and eclipse.

Date:

April, 12 2005

Software:

Eclipse 3.x

Junit 2.x

Source code:

http://www.lalilluna.de/assets/tutorials/junit-testing-source.zip

PDF Version

http://www.lalilluna.de/assets/tutorials/junit-testing-en.pdf

What is JUnit

JUnit is a simple open source Java testing framework used to write and run repeatable automated tests. It is an instance of the xUnit architecture for unit testing framework. Eclipse supports creating test cases and running test suites, so it is easy to use for your Java applications.

JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Graphical and textual test runners

What is a test case

A test case is a class which holds a number of test methods. For example if you want to test some methods of a class *Book* you create a class *BookTest* which extends the JUnit *TestCase* class and place your test methods in there.

How you write and run a simple test

1. Create a subclass of TestCase:

```
public class BookTest extends TestCase{
	//..
}
```

2. Write a test method to assert expected results on the object under test:

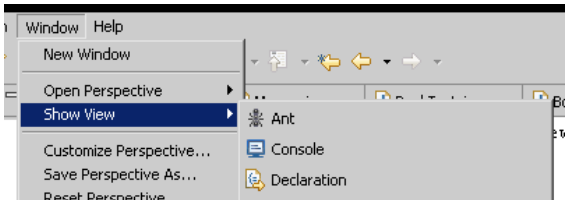
Note: The naming convention for a test method is testXXX()

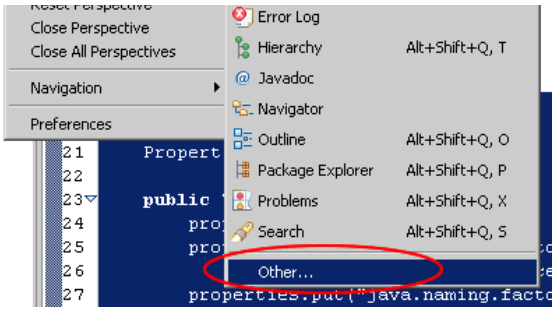
```
public void testCollection() {
	Collection collection = new ArrayList();
	assertTrue(collection.isEmpty());
}
```

3. Write a *suite()* method that uses reflection to dynamically create a test suite containing all the *testXXX()* methods:

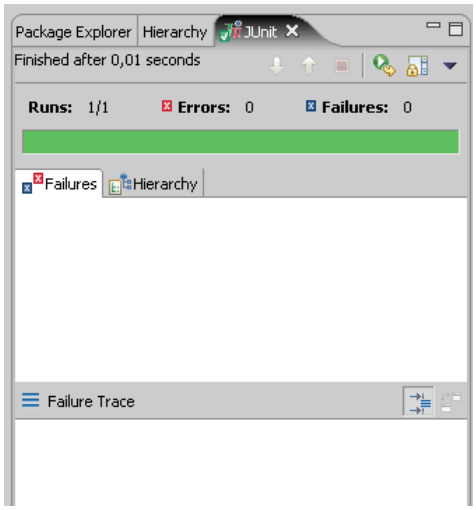
```
public static Test suite(){
	return new TestSuite(BookTest.class);
}
```

4. Activate the JUnit view in Eclipse (*Window* > *Show View* > *Other..* > *Java* > *JUnit*).





You find the *JUnit* tab near the *Package Explorer* tab. You can change the position of the tab by drag and drop it.



5.

Right click on the subclass of *TestCase* and choose *Run > JUnit Test* to run the test.

Using a test fixture

A test fixture is useful if you have two or more tests for a common set of objects. Using a test fixture avoids duplicating the test code necessary to initialize and cleanup those common objects for each test.

To create a test fixture, define a *setUp()* method that initializes common object and a *tearDown()* method to cleanup those objects. The JUnit framework automatically invokes the *setUp()* method before a each test is run and the *tearDown()* method after each test is run.

The following test uses a test fixture:

```
public class BookTest2 extends TestCase {

    private Collection collection;

    protected void setUp() {
       collection = new ArrayList();
    }

    protected void tearDown() {
         collection.clear();
    }

    public void testEmptyCollection(){
         assertTrue(collection.isEmpty());
    }
}
```

Dynamic and static way of running single tests

JUnit supports two ways (static and dynamic) of running single tests.

In static way you override the *runTest()* method inherited form TestCase class and call the desired test case. A convenient way to do this is with an anonymous inner class.

Note: Each test must be given a name, so you can identify it if it fails.

```
TestCase test = new BookTest("equals test") {
        public void runTest() {
        testEquals();
      }
};
```

The dynamic way to create a test case to be run uses reflection to implement *runTest*. It assumes the name of the test is the name of the test case method to invoke. It dynamically finds and invokes the test method. The dynamic way is more compact to write but it is less static type safe. An error in the name of the test case goes unnoticed until you run it and get a *NoSuchMethodException*. We leave the choice of which to use up to you.

```
TestCast test = new BookTest("testEquals");
```

What is a TestSuite

If you have two tests and you'll run them together you could run the tests one at a time yourself, but you would quickly grow tired of that. Instead, JUnit provides an object *TestSuite* which runs any number of test cases together. The suite method is like a main method that is specialized to run tests.

Create a suite and add each test case you want to execute:

```
public static void suite(){
        TestSuite suite = new TestSuite();
        suite.addTest(new BookTest("testEquals"));
        suite.addTest(new BookTest("testBookAdd"));
        return suite;
}
```

Since JUnit 2.0 there is an even simpler way to create a test suite, which holds all testXXX() methods. You only pass the class with the tests to a TestSuite and it extracts the test methods automatically.

Note: If you use this way to create a TestSuite all test methods will be added. If you do not want all test methods in the TestSuite use the normal way to create it.

Example:

```
public static void suite(){
        return new TestSuite(BookTest.class);
}
```

A little example

Create a new Java project named JUnitExample.

Add a package *de.lalilluna.tutorial.junitexample* where you place the example classes and a package *test.lalilluna.tutorial.junitexample* where you place your test classes.

The class Book

Create a new class *Book* in the package *de.lalilluna.tutorial.junitexample*.

Add two properties *title* of type *String* and *price* of type *double*.

Add a constructor to set the two properties.

Provide a getter- and setter-method for each of them.

Add a method trunk for a method *equals(Object object)* which checks if the object is an instance of the class Book and the values of the object are equal. The method return a boolean value.

Note: Do not write the logic of the *equals(..)* method, we do it after finish creating the test method.

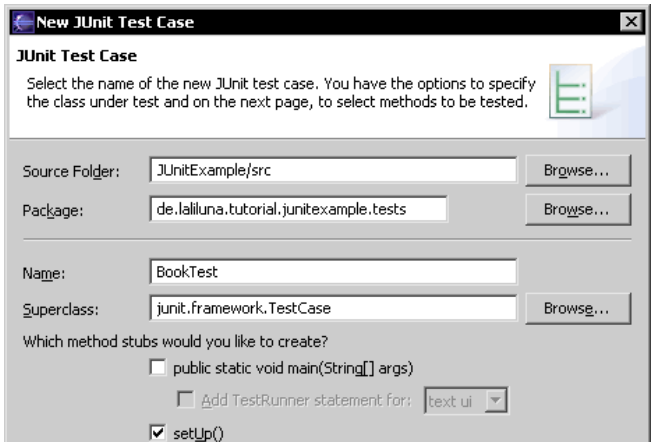
The following source code shows the class Book.

```
public class Book {

    private String title;
    private double price;

    /**
     * Constructor
     *
     * @param title
     * @param price
     */
    public Book(String title,
                double price) {
        this.title = title;
        this.price = price;
    }

    /**
     * Check if an object is an instance of book
     * and the values of title and price are equal
     * then return true, otherwise return false
     */
    public boolean equals(Object object) {

        return false;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

The test case BookTest

Create a new test case *BookTest* in the package *test.lalliluna.tutorial.junitexample* Right click on the package and choose *New > JUnit Test Case*.

In the wizard choose the methods stubs *setUp()*, *tearDown()* and *constructor()*.



The following source code shows the class BookTest

```
public class BookTest extends TestCase {

    /**
     * setUp() method that initializes common objects
     */
    protected void setUp() throws Exception {
        super.setUp();
    }

    /**
     * tearDown() method that cleanup the common objects
     */
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /**
     * Constructor for BookTest.
     * @param name
     */
    public BookTest(String name) {
        super(name);
    }

}
```

Now we want to write a test for the *equals(..)* method of the class *Book*. We provide three private properties, book1, book2 and book3 of type *Book*.

```
private Book book1;
private Book book2;
private Book book3;
```

Within the *setUp()* method we initializes the three properties with some values. Property book1 and book3 are the same.

```
protected void setUp() throws Exception {
        super.setUp();
      book1 = new Book("ES", 12.99);
      book2 = new Book("The Gate", 11.99);
        book3 = new Book("ES", 12.99);
}
```

Within the tearDown() method we cleanup the properties:

```
protected void tearDown() throws Exception {
        super.tearDown();
      book1 = null;
      book2 = null;
        book3 = null;
}
```

Now, add a test method *testEquals()* to the test case. Within the method we use the *assertFalse()* method of the JUnit framework to test if the return-value of the equals(..) method is false, because book1 and book2 are not the same. If the return-value is false the logic of the equals() method is correct, otherwise there is a logical problem while comparing the objects. We want to test if the method compares the objects correctly by using the assertTrue() method. Book1 and Book3 are the same, because both are an instance of the class Book and have the same values.

The following source code shows the testEquals() method:

```
public void testEquals(){
        assertFalse(book2.equals(book1));
        assertTrue(book1.equals(book1));
}
```

Writing the logic of the equals() method

We have finished the test and now we can add the logic to the *equals()* method stub. Open the class Book and add the logic to the *equals()* method. First we check if the object given by the method is an instance of *Book*. Then compare the properties *title* and *price*, if they are equal return true.

```
public boolean equals(Object object) {
        if (object instanceof Book) {
           Book book = (Book) object;
           return getTitle().equals(book.getTitle())
                   && getPrice() == book.getPrice();
      }
      return false;
}
```

Create the suite() method

In order to run the test method *testEquals()* add a method *suite()* to the class *BookTest*.

Note: You can also create a separate class where you add the *suite()* method.

Within the method create a new instance of *TestSuite* and use the method *addTest(..)* to add a test. Here we use the dynamically way to add a test to a TestSuite.

The method looks like the follows:

```
public static Test suite(){
      TestSuite suite = new TestSuite();
      suite.addTest(new BookTest("testEquals"));
        return suite;
}
```

Run the test

After finishing all test methods we want to run the JUnit test case. Right mouse button on the class BookTest and choose *Run As > JUnit Test*.

On the JUnit view (Menu Windows -> show view) of Eclipse you can see how many runs, errors and failures occurred.