TestExpert v0.1.0 Release Notes

General Remarks

This is the first release of TestExpert. It is a newly, beta-release of the product.

New Features

In fact everything is new here. :-)

Improvements/Enhancements

Performance Improvements/Enhancements

None at this time

Other Improvements/Enhancements

None at this time

Additional Support

jMockit and EasyMock are now fully supported. See ROADMAP for more in the future support for mocking frameworks.

Bug Fixes

Since the product is not completed yet some bugs may arise. A short bug list which **might appear** when using TestExpert is mentioned below for convenience:

As of 2014, May 23th, there were 5 verified issues that have been resolved:

Sometimes an import of an innerclass contains an invalid $

A variable which is not in the Fixture file should be treated as a literal and NOT as a variable in the testclass

Although TestExpert supports jMockit some issues may rise

There might be some security issues when a class can not be opened by TestExpert

Array types in the in- and out variable are not fully compliant. You have to change the generated class yourselve, which is trivial (2 seconds)

Known Issues

TestExpert is not compliant with the Google Web Toolkit (GWT).

Some classes are not automatically imported in the generated unittest classes. Import this classes through Ctrl-O (Eclipse).

Sometimes the type of String and int are confused. This can be fixed be changing the code in the generated test class. It will be fixed in the next release.

Jad should be installed on the developer’s pc. Jad is a Java Disassembler which disassemble binairy files to sourcecode. It should be installed on the machine's path to execute correctly.

The jMockit.jar should appear **above** the Junit 4.8.2 jar in the order / export of the jars, especially when

this message occurs when you are running jMockit as your mocking framework:

```
WARNING: JMockit was initialized on demand, which may cause certain tests to fail;
please check the documentation for better ways to get it initialized.
```

RoadMap

Near future

More successcase per method (max. one at this moment)

More unsuccessful cases per method (zero at this moment)

Far Future

Jad should not be a requirement. Will introduce a parser-generator.

Installation guide

See the INSTALL file

Optional: install the content from the demo directory in your Eclipse environment.

Getting Started Guide

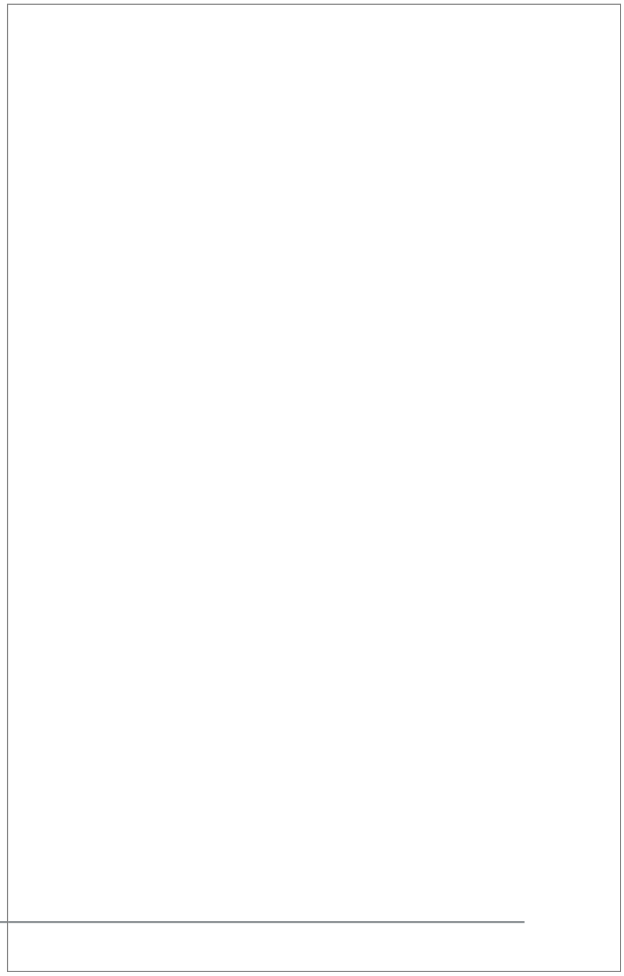
After completing the installation guide above TestExpert is ready for your first experience.

In a Java class with a method wherefore you would like to create some unittests open the desbetreffende method.

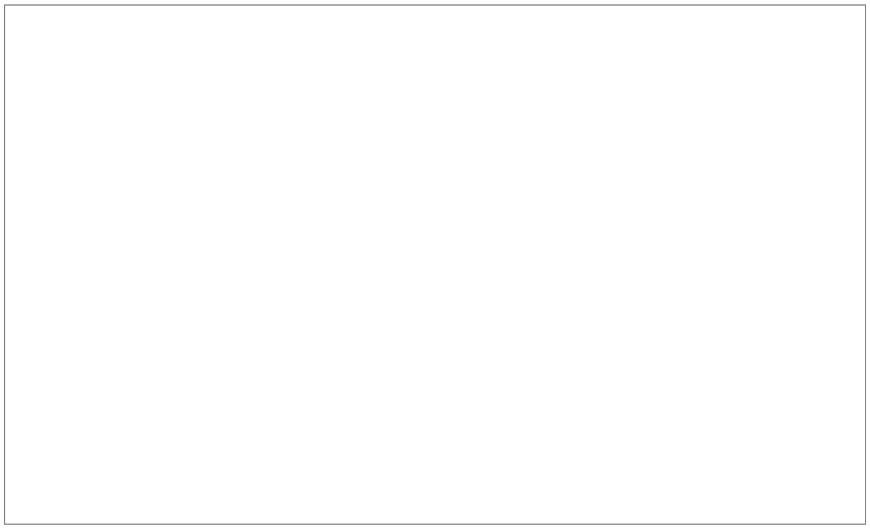
There a two ways to make a hint for TestExpert to create a unittest:

1. Use a literal: this is only possible for an int and a String

2. Use a variable: first create a so called Fixture file / class e.g.



After creating a file like this you can use the method names as a variable in your class under test WITHOUT the parenthesis! So ‘simpleDemo()’ will be ‘simpleDemo’ and ‘customer’ will be ‘customer’

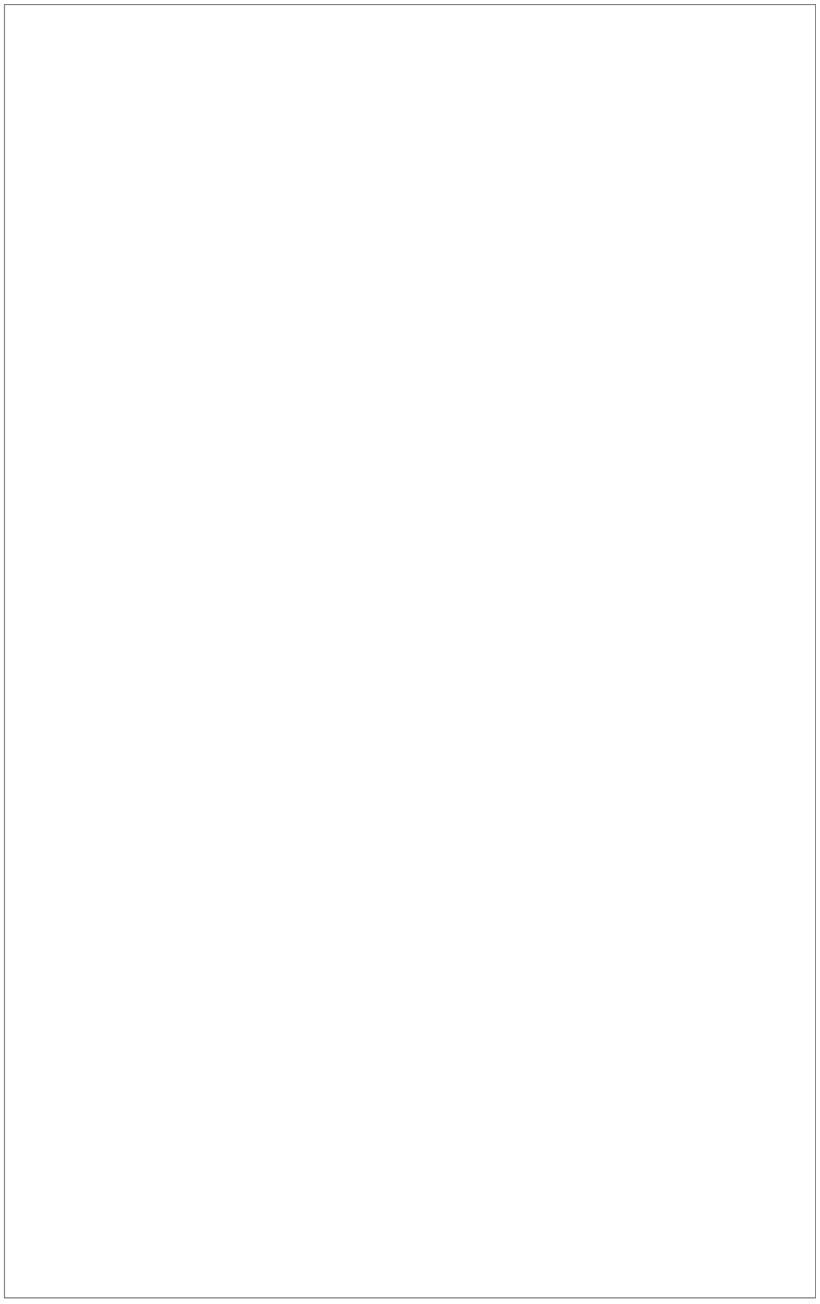


After added the @CreateUnittest annotations – the heart of TestExpert – it is time for the last step to generate your tests:

Create a class which extends from TestExpert. Let Eclipse or your favourite IDE implement the stubs for the needed template methods.

Implement the the template methods to your situation:

The methods of the overridden class contains several overriden methods which are explained below:



String getSourceFolder(): should return a String where the source files of your project are found. (TestExpert will use this a startpoint and will recursively traverse all subdirectory)

Class getFixture: should return a class which represents your Fixtures. See example above.

boolean overwriteExistingFiles: should return a boolean which indicates whether TestExpert should overwrite previously created testcases. In fact: when TestExpert has created them you may tweak and update them manually. Then you might return a false.

String getBinaryFolder(): should return the folder in which the compiled (.class) files exist. This is necessary for inspecting the JVM instructions for generating collaborating methods calls and such stuff.

String getOutputFolder(): the folder in which TestExpert will create the generated Testclasses. You should beforehand create that folder and add that folder to the sourcepath of your Java project.

String getTestsuiteName(): a suiteable name for the class which will also be generated after the creation of all your testclasses. It will contains calls to the by TestExpert generated Testclasses so you don't have to call them all by hand in your Eclipse / IDE environment.

MockFramework getMockFramework(): should return an enum instance from the MockFramework enum. At this moment only EasyMock (thoroughly tested) and jMockit (good tested) are supported.

==========

And now for the nice part and where TestExpert wins:

Writing unittests is simple for a method which does not use a collaborating class.

When you have to mock a collaborating class the test most of the time gets pretty hairy and you have to investigate more time which is counterproductive to testing and might

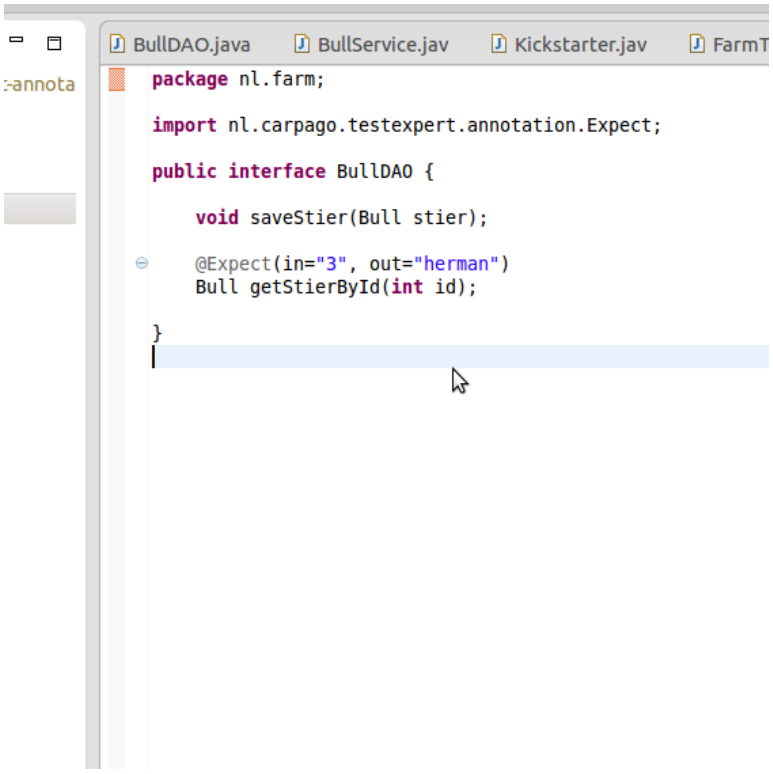
restrict yourself in harvesting good tests. Hence, of course, TestExpert can do that

tedious work for you. All you have to do is use the same paradigm as for creating

a test in your collaborating class or interface but use @Expect instead of @CreateUnittest

above the method which is called while running the code of your class under test.

For example:



After implementing this class above, we are ready to launce this as a Junit unittest.

The testclasses will now be generated in the src/test/generated-test folder (the returned String in getOutputFolder).

Have fun!

Stay informed about Carpago’s TestExpert

http://github.com/carpago/testexpert