EMMA Code Coverage

Note: in this guide only the EMMA plugin for Netbeans was implemented.

Requirements

- Netbeans
- JUnit (tests written)
- Java Application Project (not web project)

Installation

1. Download from: http://plugins.netbeans.org/plugin/38945/unit-tests-code-coverage-plugin-updated-for-netbeans-7-0
2. Extract Files
3. In Netbeans:

- - Select tools -> plugins
  - Go to Downloaded tab
  - Select Add Plugins...
  - Navigate to and select 2 extracted files, click OK
  - Select Install
  - Close plugin window

Running Tests and Viewing Report

1. Right click project -> Coverage -> Activate Code Coverage
2. Right click projet -> Test (this will run all JUnit tests in test package)
3. Right click project-> Coverage -> Show Project Coverage Statistics
4. View coverage report.

Example Run

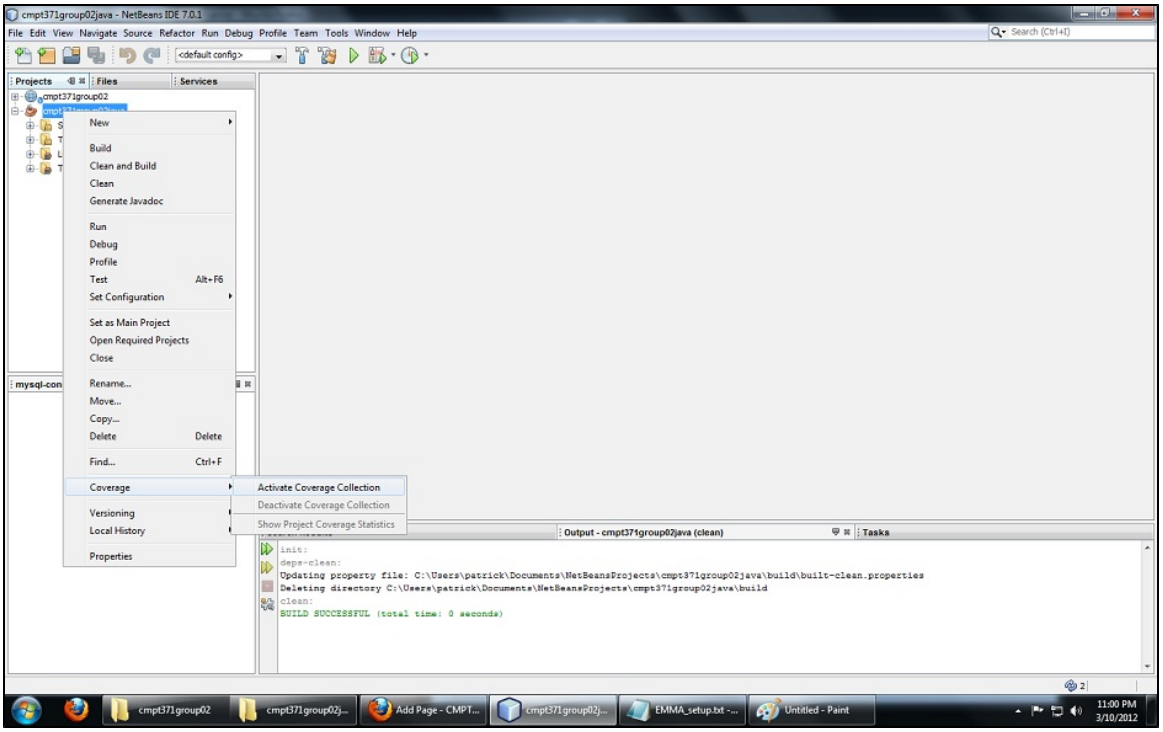
First a new Java application project was created in Netbeans.

Each Java source package and test package was copied into the new project.

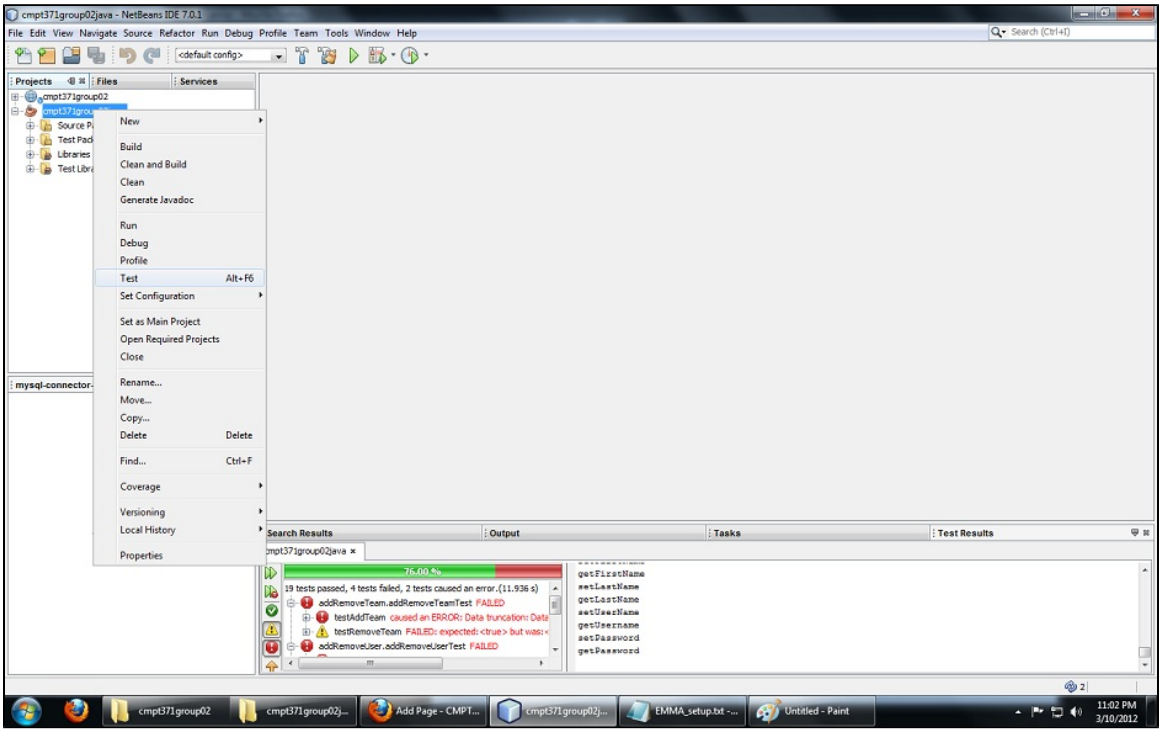
Servlets were deleted because the project setup said they had errors, and they are not unit tested anyways.

The two jar files in the library were imported into the new projects library.

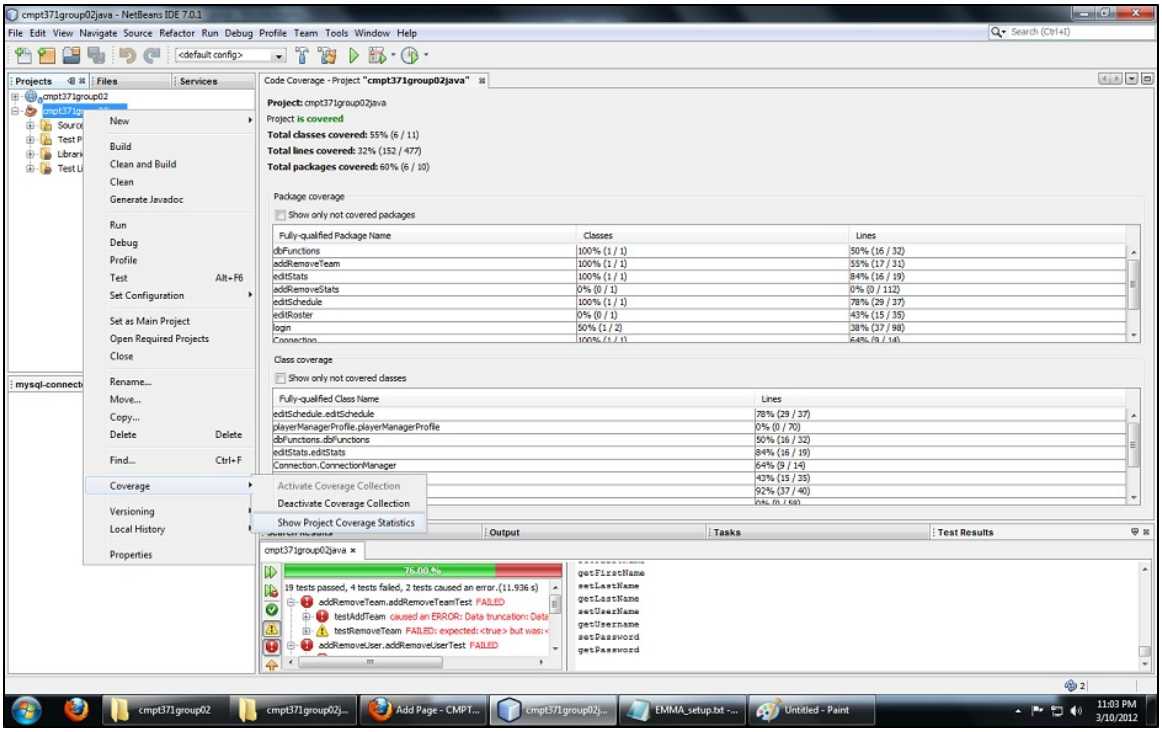
1. Activate Code Coverage



2. Test



3. Coverage Report



Usage and Summary

EMMA is a tool which seems to be very good at identify the actual coverage of any sort of tests done on a piece of software. After working out some issues with it and Netbeans and JUnist, and getting them all working together, it was fairly easy to install and implement. In Netbeans it appears to work excellently with Java applications which have implemented JUnit tests, and could potentially generate some genuinely useful data. It was simple as a activating it, running your tests, and viewing the coverage report generated. The report gives very detailed information about which packages, classes, and lines are actually hit during test execution, and also some estimation of whether or not coverage was sufficient for testing purposes, although by the looks of a generated report it appears to just check for 50% coverage.

Unfortunately this tool was not overly useful for this particular project. First of all since it is a web application, and not a Java application, it is not even possible to use EMMA on the project with out ripping it apart and mashing together a Java application with what is left. For this project that meant getting rid of all web pages such as JSP's, Javascript libraries, style sheets, and Servlets, despite the fact that Servlets are just Java code. Fortunately this leaves plain Java classes and the JUnit tests that apply to them. This method means that whenever a code coverage report is desired on current code, all of the appropriate files have to be copied into a Java project, and it also means very little of the project is actually tested since it is a web project with more than half of its content being inappropriate for this tool.

Using this method and running it on the current project (Mar. 10, 2012) generated the following results: package coverage: 60%, class coverage: 55%, line coverage: 32%. The plugin determined from these results that the project is covered. The coverage report is basically a rating of the unit test, and as it only knows about plain Java classes does not really show the whole picture. Even if that was the whole project though, it seems as if just over half of the classes and packages does not really mean that the project is covered well enough yet.

One note about this tool is that it does not have to be used as a plugin for Netbeans. It can be used, and was briefly used, from the command line on JAR files. In this manner EMMA can be used to show the code coverage of regular usage rather than the coverage of unit testing. This does seem to work fairly well, however in the very brief time spent with it in this way, it did seem to have trouble dealing with multi-threaded applications. This project was never run in this way since it can't actually run outside of unit testing without the web pieces.