**Code Coverage Tools Overview**

**Emma** vs **Clover** vs **Cobertura’s**

Contents

[Common Ratting 1](#_Toc361053435)

[Maven Plugin (Cobertura) 1](#_Toc361053436)

[Emma overview 2](#_Toc361053437)

[**Features** 3](#_Toc361053438)

[**Launching** 3](#_Toc361053439)

[**Analysis** 3](#_Toc361053440)

[**Import/Export** 4](#_Toc361053441)

[Code Coverage Tools (Clover, Cobertura, Emma) Comparison 4](#_Toc361053442)

[Units of coverage 4](#_Toc361053443)

[Eclipse plugin 4](#_Toc361053444)

[Full report 5](#_Toc361053445)

[Static nested classes 6](#_Toc361053446)

[Enums 7](#_Toc361053447)

[Annotations 7](#_Toc361053448)

[Default no-arg constructor 7](#_Toc361053449)

[Conclusion 7](#_Toc361053450)

# Common Ratting

|  |  |  |  |
| --- | --- | --- | --- |
| Plugin | Emma (base on JaCoCO) | Maven Plugin (Cobertura) | **Clover** |
| Eclipse support | Y | Y | Y |
| License | CPL | |  |  | | --- | --- | |  |  |   GNU GPL | [Proprietary](http://en.wikipedia.org/wiki/Proprietary_software) |
| highlight covered and uncovered code | Y | **Y** | Y |
| **Eclipse plugin (**nice graphic display and) let you drill down | Yes | **Yes** | Weak, no drill-down into classes and no visual graph of coverage |
| **Full report(**nice graphics**)** | **strong** | **strong** | stronger |
| reports include metrics | **strong** | **strong** | Stronger |
| **catch for Static nested classes** | **Y** | **N** | N |
| **catch for Enums test** | **Y** | **N/A** | N |
| **catch for Annotations** | **N** | **N/A** | N |
| **catch for Default no-arg constructor** | **y** | **N/A** | N |
| **Ratting** | **4.5** | **4.5** | **5** |

# Emma overview

EclEmma is a free Java code coverage tool for [Eclipse](http://www.eclipse.org), available under the [Eclipse Public License](http://www.eclemma.org/license.html). It brings code coverage analysis directly into the Eclipse workbench:

* **Fast develop/test cycle:** Launches from within the workbench like JUnit test runs can directly be analyzed for code coverage.
* **Rich coverage analysis:** Coverage results are immediately summarized and highlighted in the Java source code editors.
* **Non-invasive:** EclEmma does not require modifying your projects or performing any other setup.

Since version 2.0 EclEmma is based on the [JaCoCo](http://www.eclemma.org/jacoco) code coverage library. The Eclipse integration has its focus on supporting the individual developer in an highly interactive way. For automated builds please refer to [JaCoCo documentation](http://www.eclemma.org/jacoco/trunk/doc) for [integrations with other tools](http://www.eclemma.org/jacoco/trunk/doc/integrations.html).

[Originally](http://www.eclemma.org/installation1x.html) EclEmma was inspired by and technically based on the great [EMMA](http://emma.sourceforge.net/) library developed by Vlad Roubtsov.

The [update site](http://www.eclemma.org/installation.html) for EclEmma is **http://update.eclemma.org/**. EclEmma is also available via the Eclipse [Marketplace Client](http://marketplace.eclipse.org/marketplace-client-intro), simply search for "EclEmma".

**Features**

**Launching**

EclEmma adds a so called *launch mode* to the Eclipse workbench. It is called *Coverage* mode and works exactly like the existing *Run* and *Debug* modes. The *Coverage* launch mode can be activated from the *Run* menu or the workbench's toolbar:

Launching Toolbar

Simply [launch](http://www.eclemma.org/userdoc/launching.html) your applications or unit tests in the *Coverage* mode to collect coverage information. Currently the following launch types are supported:

* Local Java application
* Eclipse/RCP application
* Equinox OSGi framework
* JUnit test
* TestNG test
* JUnit plug-in test
* JUnit RAP test
* SWTBot test
* Scala application

**Analysis**

On request or after your target application has terminated code coverage information is automatically available in the Eclipse workbench:

* **Coverage overview:** The [*Coverage* view](http://www.eclemma.org/userdoc/coverageview.html) lists coverage summaries for your Java projects, allowing drill-down to method level.
* **Source highlighting:** The result of a coverage session is also directly visible in the Java source editors. A customizable [color code](http://www.eclemma.org/userdoc/annotations.html) highlights fully, partly and not covered lines. This works for your own source code as well as for source attached to instrumented external libraries.

Additional features support analysis for your test coverage:

* **Different counters:** Select whether instructions, branches, lines, methods, types or cyclomatic complexity should be summarized.
* **Multiple coverage sessions:** Switching between coverage data from multiple [sessions](http://www.eclemma.org/userdoc/sessions.html) is possible.
* **Merge Sessions:** If multiple different test runs should be considered for analysis coverage sessions can easily be merged.

**Import/Export**

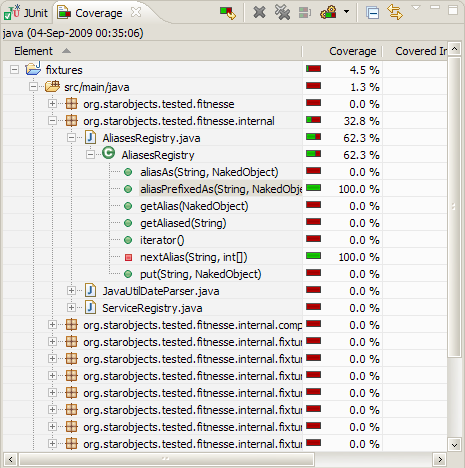
While EclEmma is primarily designed for test runs and analysis within the Eclipse workbench, it provides some import/export features.

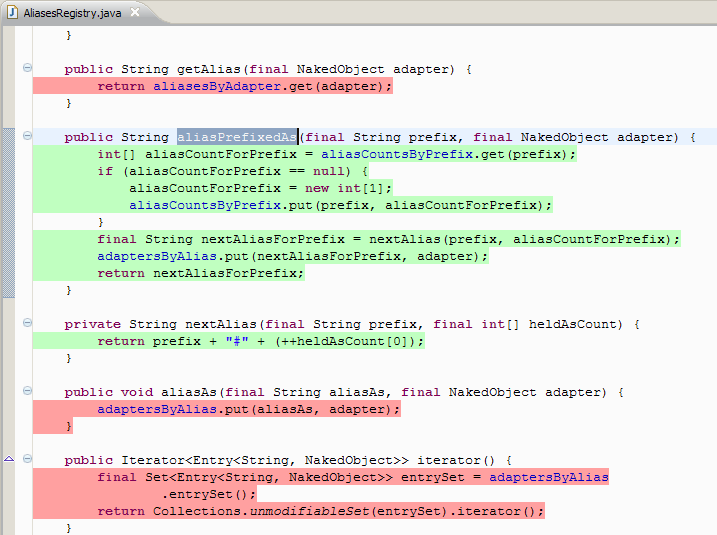
* **Execution data import:** A wizard allows to [import](http://www.eclemma.org/userdoc/importexport.html) JaCoCo \*.exec execution data files from external launches.
* **Coverage report export:** Coverage data can be [exported](http://www.eclemma.org/userdoc/importexport.html) in HTML, XML or CSV format or as JaCoCo execution data files (\*.exec).
* **Color:**

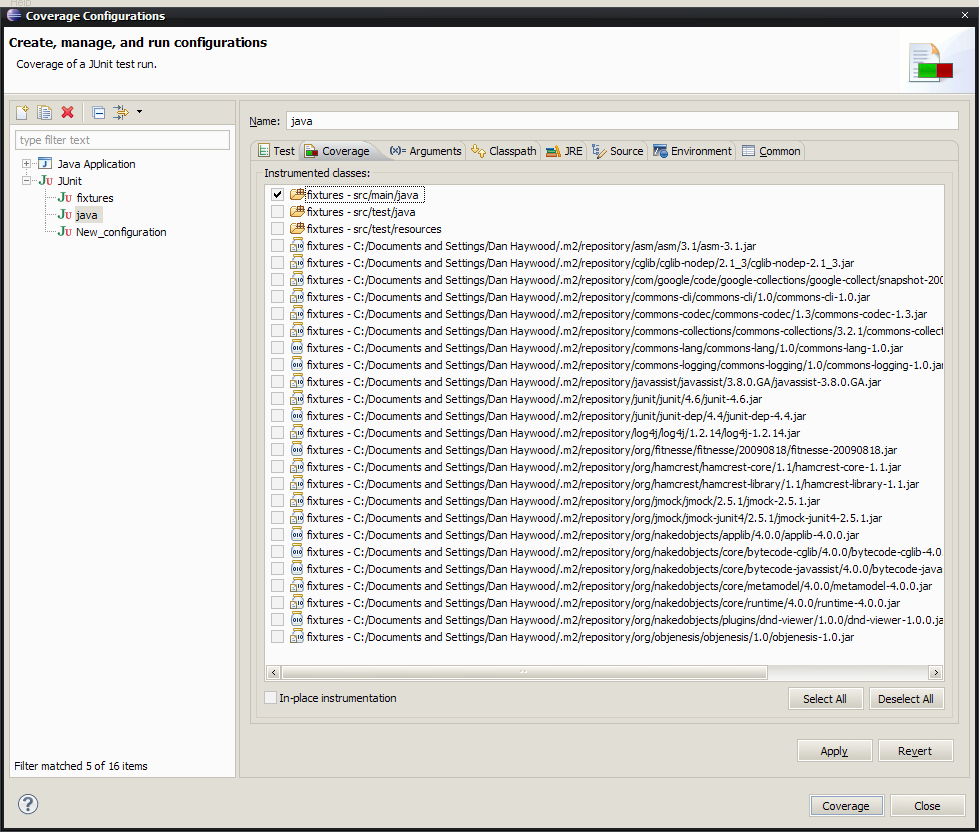
Đỏ : not coverage

Xanh: coverage

Vàng: miss cover (miss branch …)



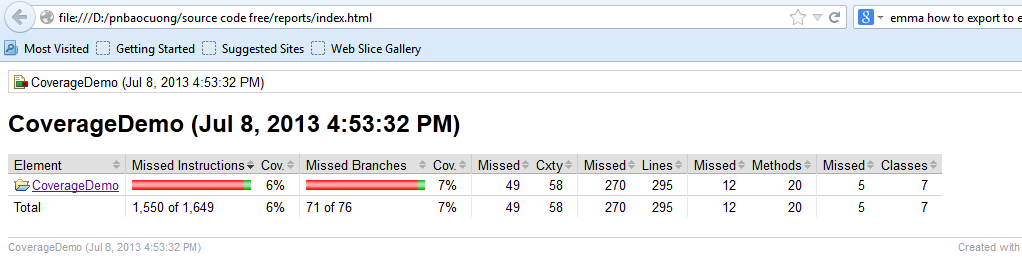




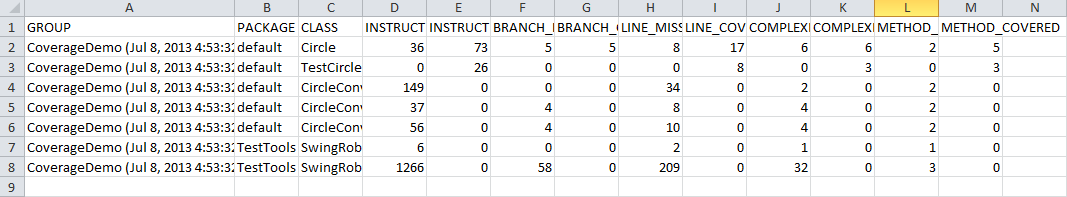
# Example picture of test coverage for Circle project

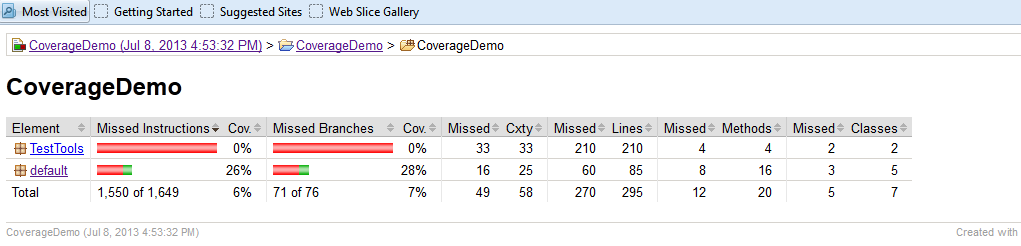
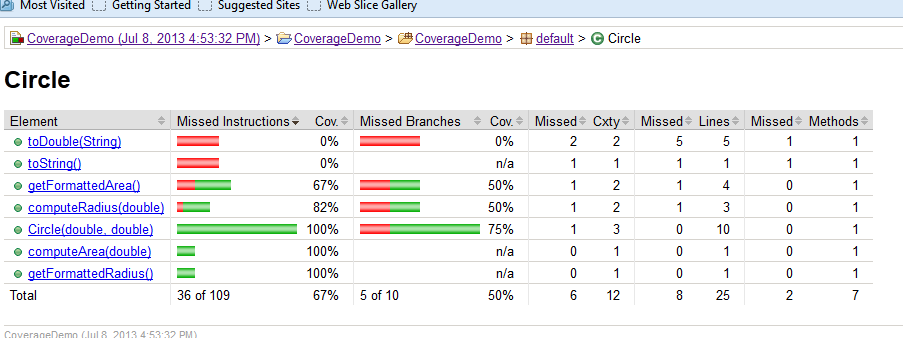
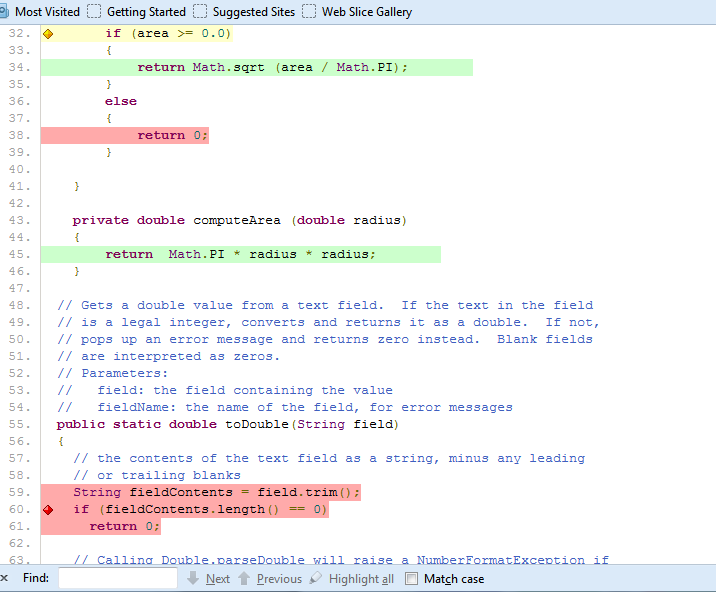
# 

* **HTML report**



* **CSV report**



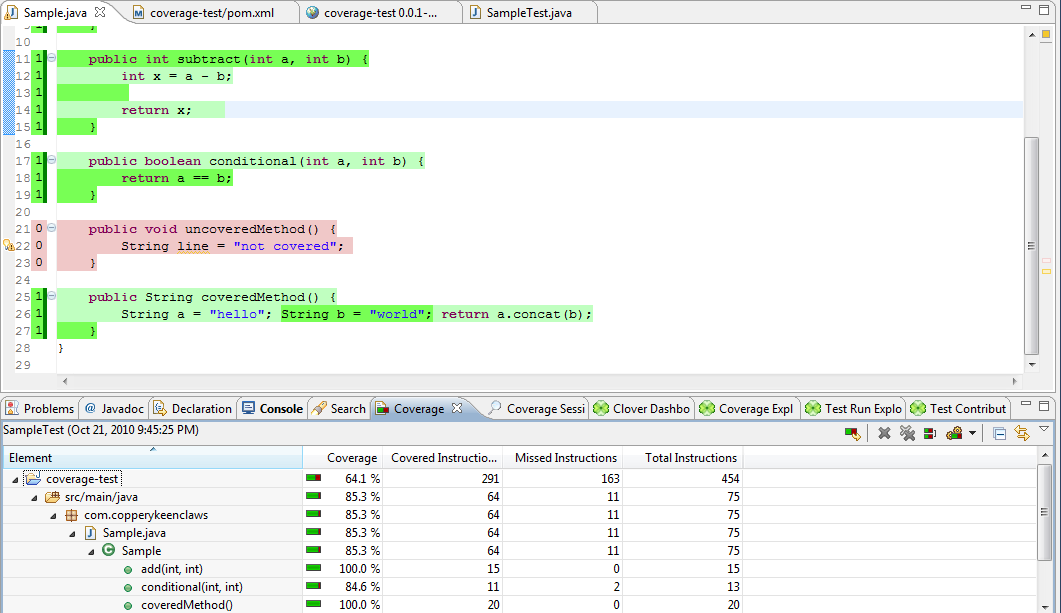
# Code Coverage Tools (Clover, Cobertura, Emma) Comparison

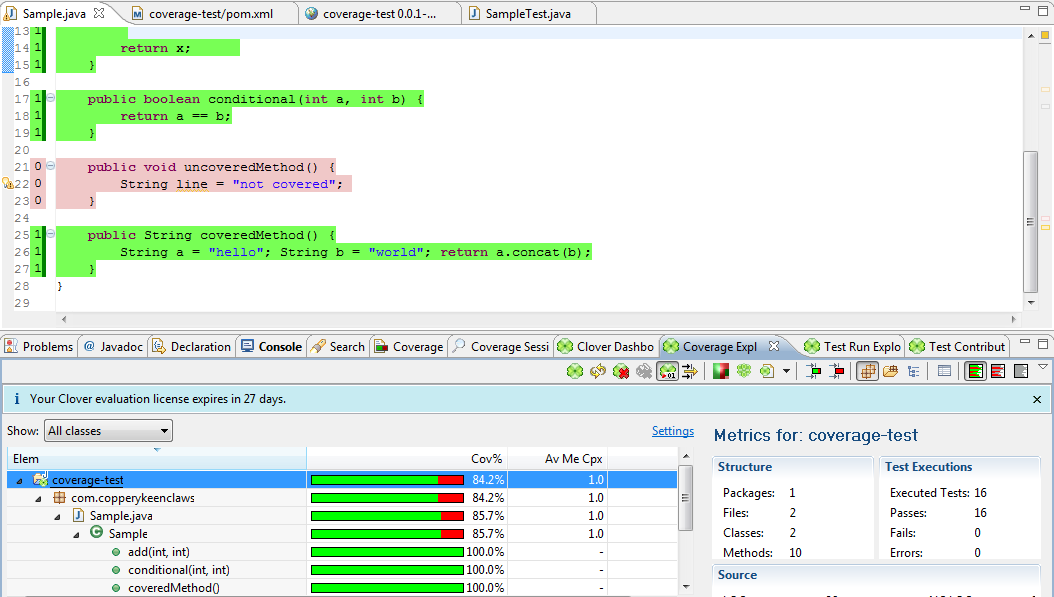
## Units of coverage

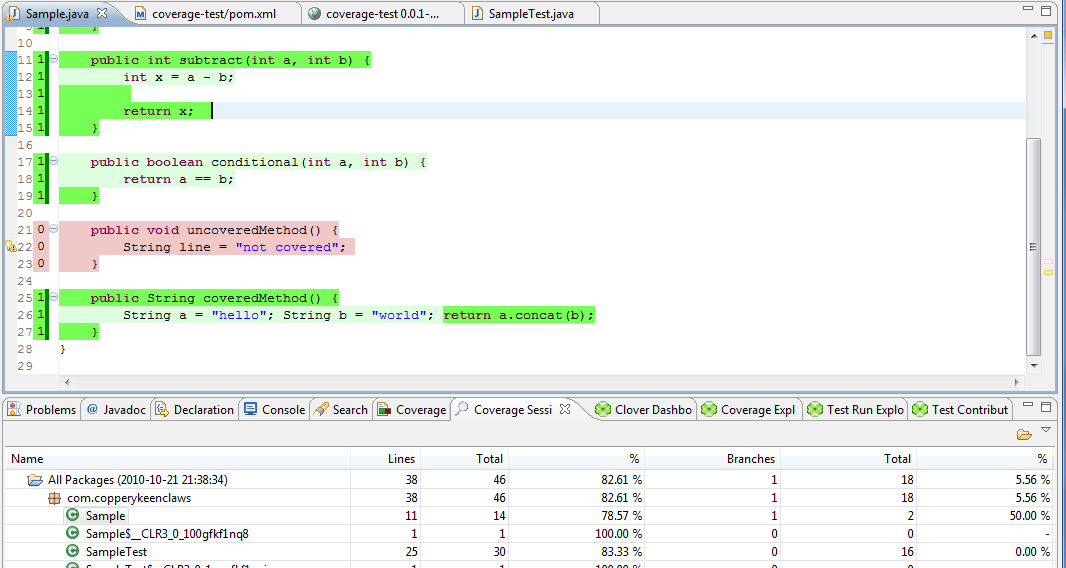
Cobertura focuses on lines and branches (ie. if and case statements); EclEmma concentrates on bytecode instructions, though you can also get line metrics. Emma has a good write-up of their [methodology](http://emma.sourceforge.net/faq.html#q.blockcoverage), but I didn’t see as much for Cobertura. Clover looks at statements, branches, and methods to come up with [a total score](http://confluence.atlassian.com/display/CLOVER/About+Code+Coverage). All these approaches result in fairly similar coverage results (unless your code does things like put multiple statements on the same line separated by semi-colons) but they’ll almost always be a little different.

## Eclipse plugin

All the tools will highlight covered and uncovered code in Eclipse for you. The Emma and Clover plugins have a nice graphic display and let you drill down.

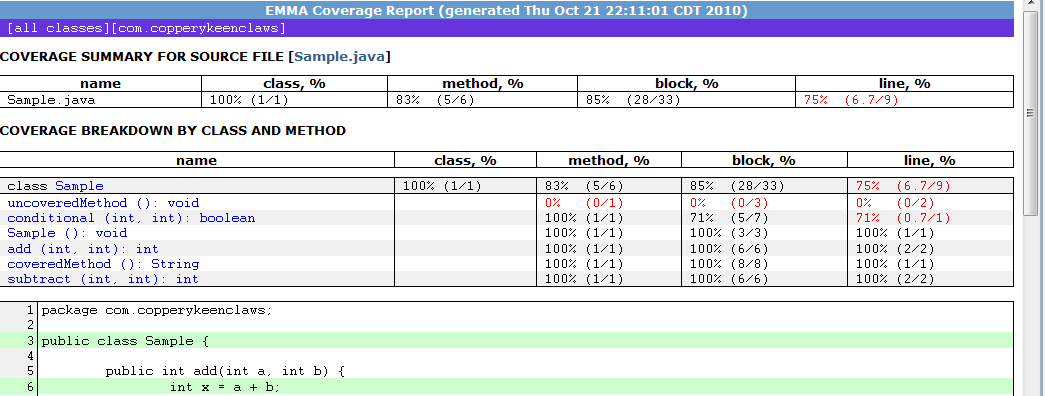
Emma:  


Clover:  


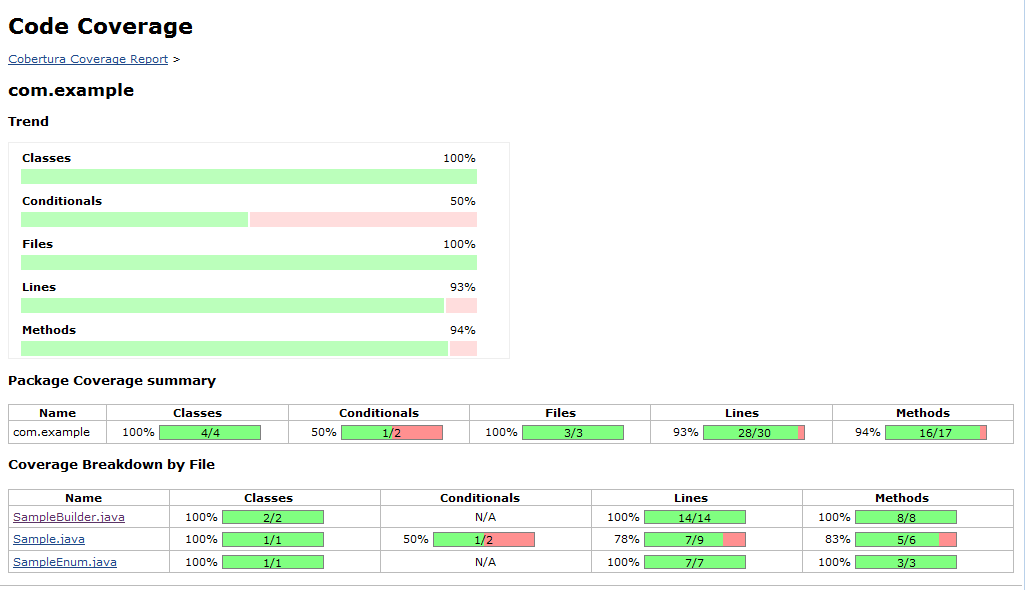
Cobertura’s display in Eclipse is a bit weak, no drill-down into classes and no visual graph of coverage:  


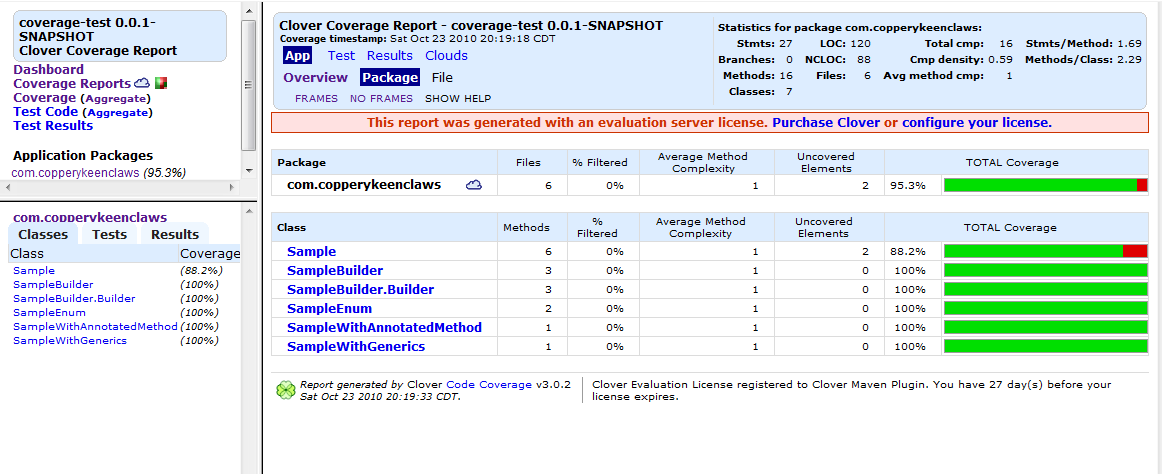
## Full report

Probably more important than the Eclipse view are the reports that each tool can create, eg. in an automated build. I didn’t take much time to explore different configuration options here, just took the defaults for each.

Emma’s report actually has less visual appeal than its Eclipse plugin. If you invest some time, you can probably make it look better but this is what you get out of the box:  


Cobertura, on the other hand, has some nice graphics. If you just configure report output as xml, and turn on the Hudson plugin, it looks like:



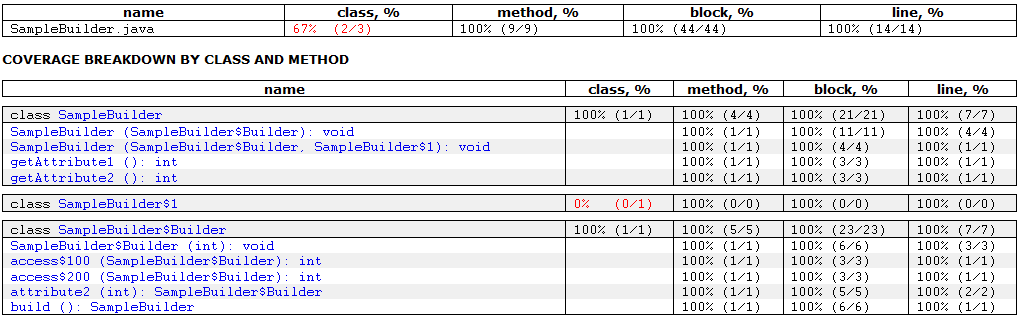
And here’s Clover:  


The Emma and Cobertura reports include metrics on classes, lines, and methods. Cobertura also has conditional statements and files while Emma includes blocks. Conditionals could be interesting, I guess, if you notice a big coverage gap. Clover’s ‘elements’ appears to be lines and methods. Cobertura has nice drill-downs and Clover has not just drill-downs but tree map and word cloud charts as well.

Aside from graphics, I also looked a little closer at how some specific code elements are handled.

## Static nested classes

The code under test for the result page below has an internal builder class. If you’ve never noticed, the compiler creates an extra $1 class for you behind the scenes. Emma reports the SampleBuilder$1 class as uncovered:



So somehow you have 100% coverage of blocks, lines, and methods, but only 67% coverage of classes… Cobertura excludes SampleBuilder$1 in its class coverage to give you 100%, but if you drill-down, you see the class with N/A in its line and method coverage columns, which seems like a good solution. Clover does not show the $1 class at all, which is also fine.

## Enums

Emma dings you if your tests don’t explicitly test the built-in enum valueOf and values() methods- they show up as 0% coverage. Like with the inner classes, Cobertura will show those methods but mark them as N/A and not include them in your method coverage percentage. Clover will not show them at all.

## Annotations

Cobertura will show you an annotation (an @interface that you’ve created) with N/A. It won’t appear at all in Emma. It also won’t appear in Clover, but curiosly, I did notice it shows up in a word cloud chart of top project risks.

## Default no-arg constructor

If your class doesn’t define its own constructor, but a test calls the implicit no-arg constructor, does it count in the coverage? For Emma, yes. It lists the constructor as classname(), eg. Sample() and gives you credit for covering one method and one line. Cobertura will show the constructor as <init>() but give it an N/A for coverage. Clover won’t show an undeclared default constructor at all.

## Conclusion

These are all good tools. In Eclipse, I’m mostly concerned with finding untested lines and getting a general sense of how much code is covered so I think I’ll keep using EclEmma. For a full report, Cobertura has a nicer look and fewer coverage oddities to note. Also, even though new plugins have come out recently for Emma, it doesn’t appear that the main project is still being actively developed. As far as speed goes, we try to keep our unit tests very fast anyway, so it was not a big concern and I didn’t bother investigating it.

Clover looks goods visually, has lots of options, and I like how it handles results. If money were no object, I’d definitely look more closely at it, but for our situation, the free tools work just fine.