Assignment 4 - Introduction to Software Engineering Tools and Environments

Davison Bullock MSc Information Technology PT Year 2 2012-2013 10th February 2013

**EclEmma v2.2.0 plug-in for Eclipse IDE Tutorial**

**What is EclEmma and how does it fit into the SDLC Lifecycle?**

EclEmma is a free to use Java code coverage tool for Eclipse IDE. It integrates JaCoCo code coverage library from v2.0 onwards and is starting to become available in tools and different development IDE's. (Ref 2). The tool allows code coverage analysis functionality to be brought directly into the Eclipse IDE workbench for ease of use (i.e. the development environment where the source code to be analysed has been developed and tested). The tool is free and available via the Eclipse Public license.

EclEmma used to be based on the popular EMMA code coverage library which is no longer supported (Ref 2).

EclEmma allows code coverage and trace information to be reported from test cases that are executed within Eclipse.

It allows the degree of testing carried out on software during the testing phase of the software development lifecycle to be determined by code coverage analysis and is used as a software testing measure (Ref 2). The measure of the completeness of set of test cases defines coverage (Ref 4)

The following main code coverage criteria are covered by EclEmma. One of more of these are used to measure how well a program is exercised by a test suite e.g. JUnit tests (Ref 3 and Ref 5).

The goal is to execute as much source code as possible from test cases. The types of code coverage that can be covered are: (Ref 3)

* **Condition coverage** - Determines if both true or false has been evaluated for a boolean expression.
* **Decision coverage**- Has every edge in a program been executed e.g. every branch in a control structure such as a java switch statement.
* **Function/Method coverage** - In the program has each function or method been called.
* **Statement coverage** - Checking that every node in a program has been executed

Examples of the above can be found at[**http://en.wikipedia.org/wiki/Code\_coverage**](http://en.wikipedia.org/wiki/Code_coverage)

**Why Use EclEmma?**

Can be run with JUnit - JUnit test cases are run by EclEmma and a coverage report is produced after the test cases have been executed. A tool such as EclEmma is useful for checking code coverage of unit test cases for large programs where calculating code coverage by hand would be tedious.

Code coverage test results can be used to help developers or testers create test cases for important functions within the source code.

Developers can interact with EclEmma from within Eclipse.

Code coverage analysis can be supported in automated builds by integrating the underlying JaCoCo library.

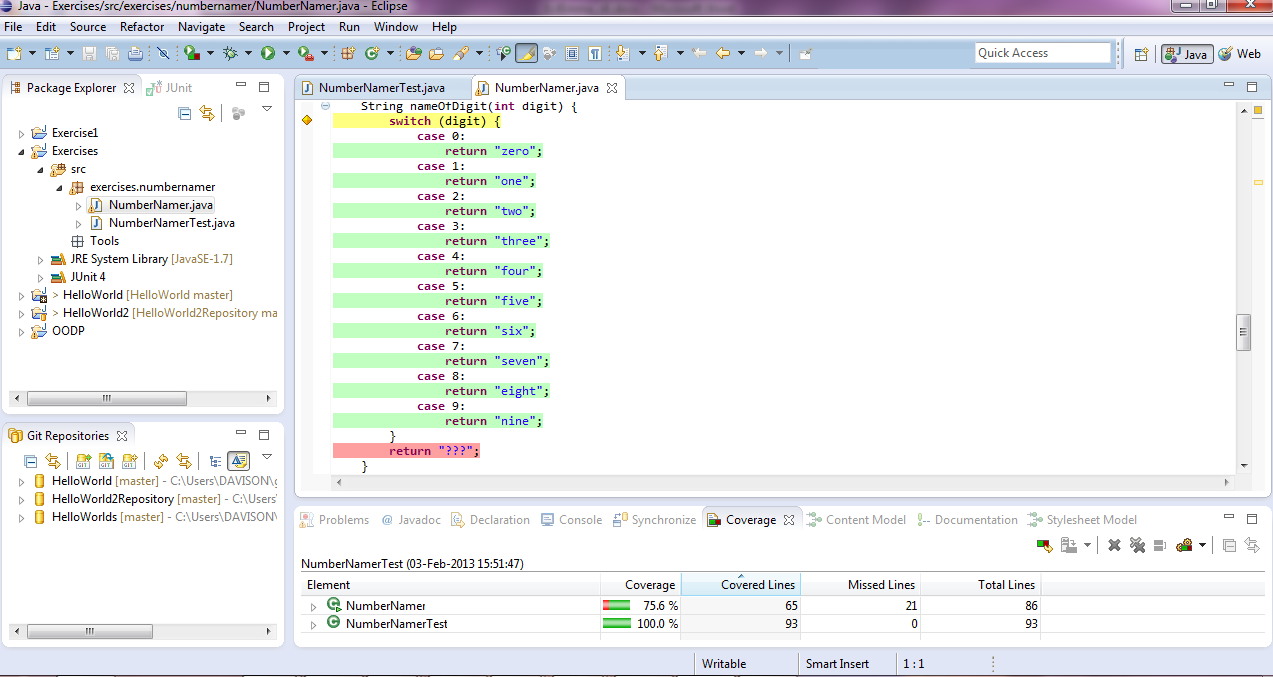
**Benefits**

EclEmma allows JUnit tests to run and be analyzed immediately for code coverage as it is launched like JUnit from the Eclipse IDE workbench.

Immediate Code Coverage results are highlighted and are available in the code coverage window.

EclEmma is non invasive. There is no need for users to modify existing java projects and to have any additional setup.

**1. Overview of Key Features**

****

* **EclEmma is based on JaCoCo (Refer to Section 6 - A little bit about JaCoCo).**
* **Non invasive -** Sourcecode does not have to change to allow code coverage to run and projects do not need to be modified.
* **Coverage statistics** - Coverage reporting is done as soon as an EclEmma coverage is run.
* **Coverage results are produced immediately** and shown in the development IDE as soon as a coverage has been run for a Launch type.
* **Launch types** - EclEmma allows coverage reports to be produced for the following launch types as applications or unit tests. The following are supported by launching in coverage mode to collect coverage information.
* Eclipse/RCP application
* JUnit test
* JUnit Plug-in Test
* JUnit RAP test
* Java application
* Scala application
* **Code Coverage Analysis**
* Involves EclEmma highlighting source code.
* EclEmma selects the statistics/measures to return from those that are available. Measures are called counters.

Counters are available for the following source code features:

* Instructions
* Branches
* Lines
* Methods
* Types
* Cyclomatic Complexity - the measure of independent paths through a programs source code (Ref 5)

Allows code coverage to be viewed for project files down to method level of detail

**1.1 Analysis**

Code coverage information is available automatically in the Eclipse workbench after the application has run or from the user manually requesting anytime to show this information.

**Source code highlighting** - Results of code coverage analysis is shown in the Eclipse IDE code editor.

**Coverage view** - Summarises the coverage(s) available in the users Java profile. Users can inspect down to the method level. Highlighting occurs on source code and any attached external libraries that are linked.

**Users can select counter types to show**. In Coverage View; types available are instructions, branches, lines methods, types or cyclomatic complexity.

**Users can switch between coverage data that is in multiple coverage sessions.**

**Multiple coverage sessions can be merged.** If the coverage analysis requires different test runs to be analysed

**1.2 Importing/Exporting**

As well as allowing users to do code coverage analysis via the Eclipse Workbench EclEmma allows Import / Export.

**Import Execution data -** JaCoCo (\*.exec) files can be imported from external applications and unit tests (launches). The user can do this via a wizard.

**Coverage Report Export -** Coverage data can be exported as JaCoCo execution data (\*.exec) or XML, HTML or CSV.

**2. Installation**

The latest release of EclEmma requires Java 1.5 run time (JRE) and higher to already be installed and that Eclipse IDE 3.5 and higher be used. The Java Development Kit (JDK) should already be installed. If it is not then this will need to be downloaded and installed before EclEmma is installed.

There are 3 ways to install EclEmma these include:

**2.1 Manual download and Installation**

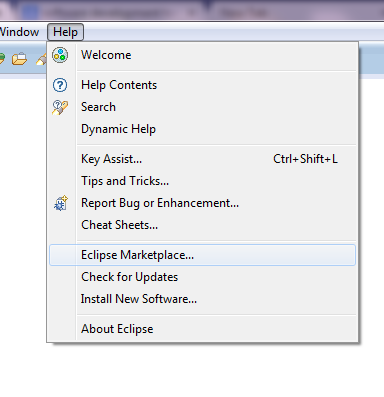
The EclEmma software can be downloaded and installed manually. The required version of the software can be downloaded from <https://github.com/jacoco/eclemma/downloads>. The software is available packaged as a zip file and has to be unzipped to the drop-ins sub folder in the Eclipse installation directory. Below is an example eclipse installation.

C:\eclipse\dropins

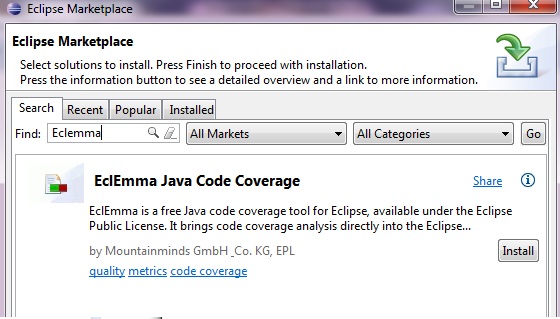
Eclipse will need to be restarted before EclEmma can be used.

**2.2 Installation from Eclipse Marketplace**

1. Select Help, then select Eclipse Marketplace



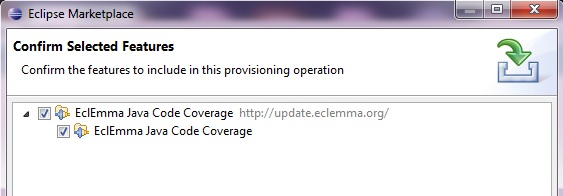
2. Type EclEmma in the Eclipse Marketplace dialog find box and select find. EclEmma should appear in the search results as below. Click install button to start installation.



3. Confirm you want to install selected software and hit next button.

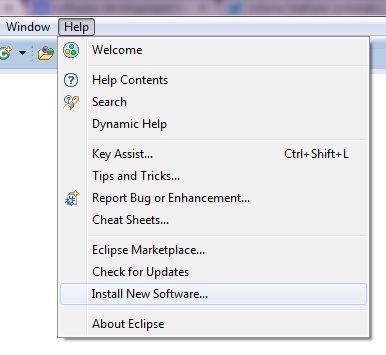
4. Review and accept licence agreement and hit Finish button.

5. EclEmma will begin installation and you will be prompted to restart Eclipse after installation.



**2.3 Installation from Eclipse Install New Software**

1. Select Help and then select Install New Software

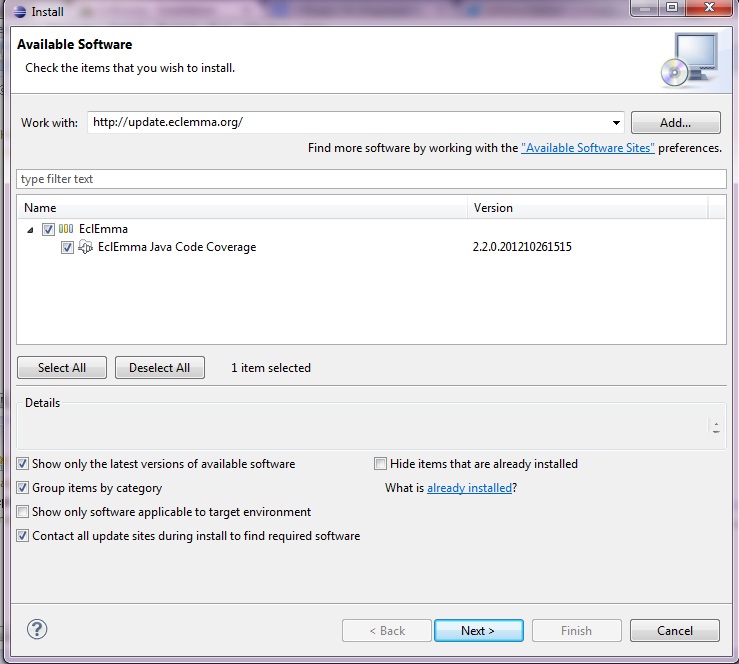
****

2. Type EclEmma in "Works with" text box in the Install dialog that appears.

3. A URL **http://update.eclemma.org/** to the installation site can be selected for the software if it exists.

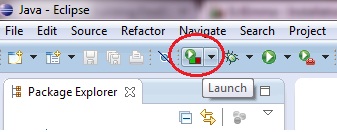
4. Confirm the tool to install by ticking it or pressing Select All button in the dialog.

5. Click Next and follow the installation as it installs.



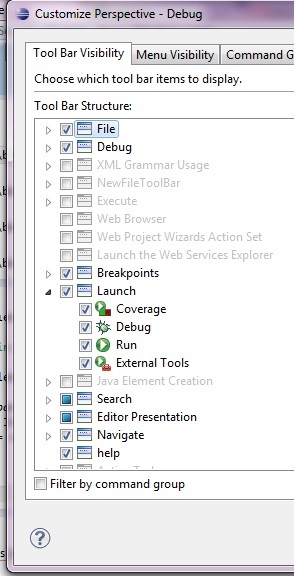
**2.4 Checking software has been installed**

1. Once EclEmma has been installed the coverage launch tool should appear in the Eclipse toolbar



**3. Running EclEmma**

Eclipse IDE allows java program to be  *run in Run* or *Debug* mode. EclEmma adds coverage mode as shown above to the Eclipse toolbar. If the coverage launch tool is not visible then it may require you to enable it via the Customize Perspective dialog accessed via the Window menu in Eclipse.



**3.1 The Code Coverage Analysis Process**

For Code Coverage analysis in EclEmma. The process always involves

1. Running the coverage

2. Analyzing coverage results

Coverage results are generally better when they are repeatable. Repeatability can be achieved by making the coverage run an automated application or tests such a unit tests as with a JUnit test case/suite.

1. Press Run Launch tool button to run a coverage.

2. Selecting a java application or unit test/suite to run

3. Code coverage will be highlighted (annotation) in the Eclipse IDE code window

4. The code coverage results (counts) are returned in the Code Coverage Window.

**4. Launching a Coverage in Coverage Mode**

This is similar to running a Java program from the Eclipse IDE workbench. The running of a program or unit test(s) to be analyzed in EclEmma is done from the Eclipse IDE workbench through a launch coverage tool/button. This is added to Eclipse Run menu and toolbar after EclEmma is installed and sits alongside the Run or Debug mode on the toolbar. Launch is the term for running the code coverage analysis on the Java program (application) or unit tests.

**Toolbar with Coverage Launch button selected**

****

The following launch types are supported

* Eclipse/RGB application
* Equinox OSGI framework
* Local Java application
* JUnit test
* TestNG test
* JUnit plug-in test
* Scala application

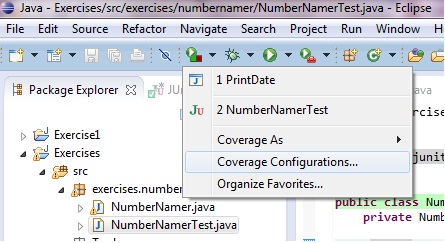
The user can choose to launch their applications or unit tests for analysis in Coverage Mode using the default settings

The user can also choose to run the Coverage from the menu, toolbar or via a context menu. The context menu (via mouse right click) lets the user choose to run Coverage As. This allow users to choose a specific Java element to launch in Coverage Mode and works similarly to the Run and Debug modes in Eclipse. The user has the same ability as those modes to change settings for the coverage launch via the Coverage Launch dialog

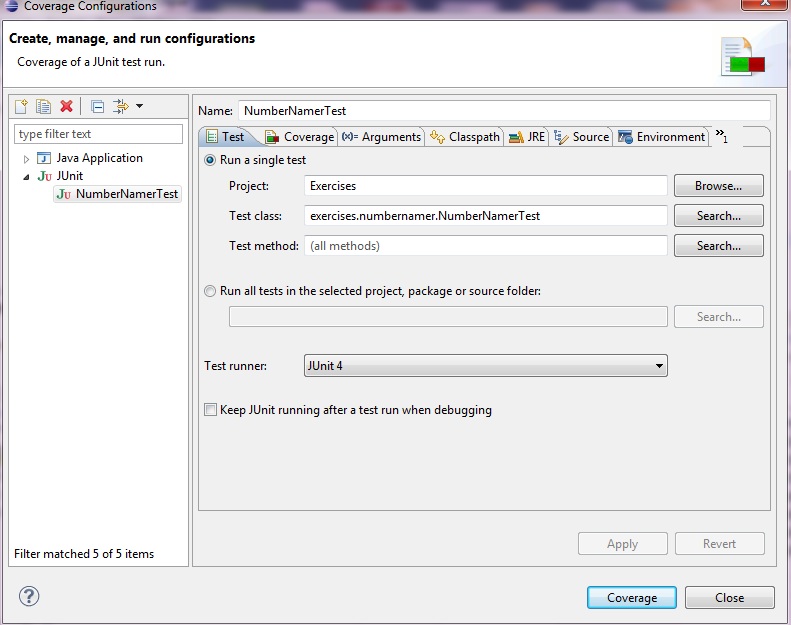
**Changing settings for Coverage Launch**

1. Go to the Coverage Launch tool and click it

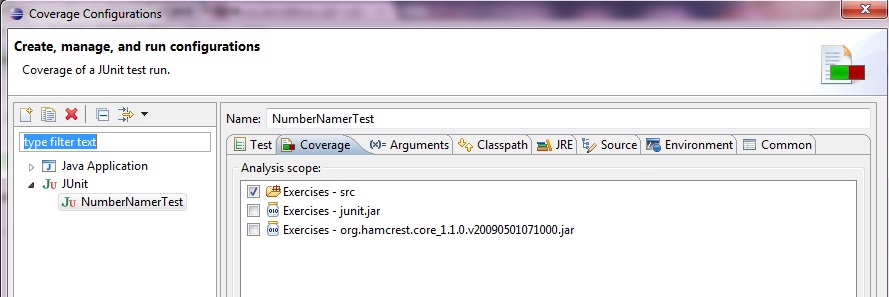
2. Select Coverage Configurations from the drop down list that is shown



3. The Coverage Configurations dialog will appear (below)

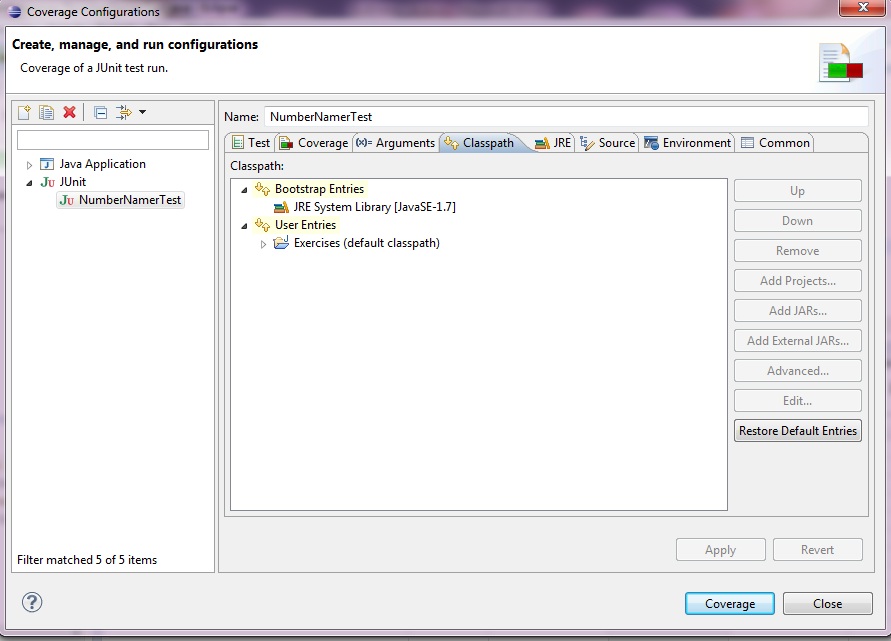


**Coverage Tab**

****

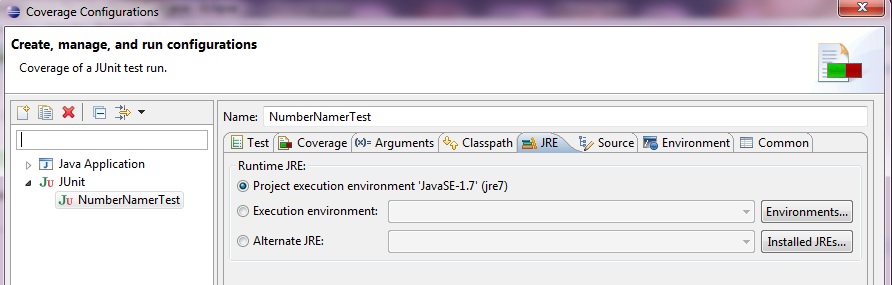
The Coverage tab allows users can choose what files and libraries to include in a coverage (scope for analysis) from those that are available (loaded in coverage session).

**Classpath tab**

****

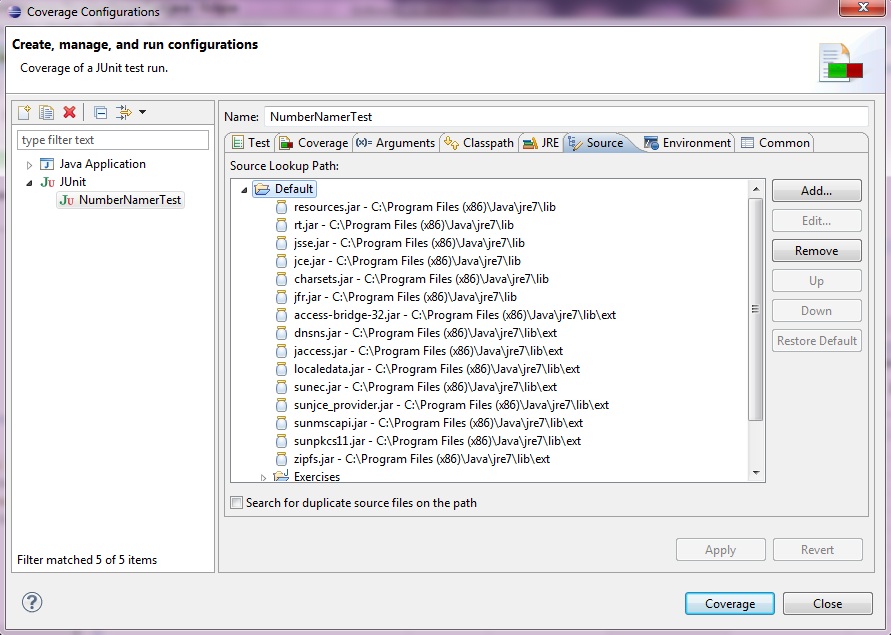
The Classpath tab above the user has to select at a least one class path entry listed in the coverage tab in order to launch an application in coverage mode. It is possible to set the rules that allows EclEmma to choose a classpath by default using the Code Coverage Preferences dialog.

**JRE tab**

****

The JRE Tab allows users to select the Java Run Time Environment to use and to see which ones are installed.

**Source Tab**

****

The source tab allows users to see what source files and libraries are part of the coverage session and they can choose to add or remove files using the add or remove buttons.

**When EclEmma finished running the coverage (application or unit tests) coverage data that is collected is automatically displayed to the user. EclEmma cannot show coverage data results if the Java VM is not running or if Eclipse is closed.**

**5. Producing a Coverage Report**

Load a java project and create some JUnit test cases or have some that were already produced before coding via (Test Driven Development (TDD)).

A github repository exists with some code to use in this tutorial and can be found at the URL below. The repository consists of 2 class files. One is called NumberNamer.java which allows users to type a string and numbers that are entered are identified and converted to its word format. The other class is a JUNIT test class with a suite of unit tests written to test the NumberNamer. java class.

<https://github.com/dbullo01/NumberNamer>

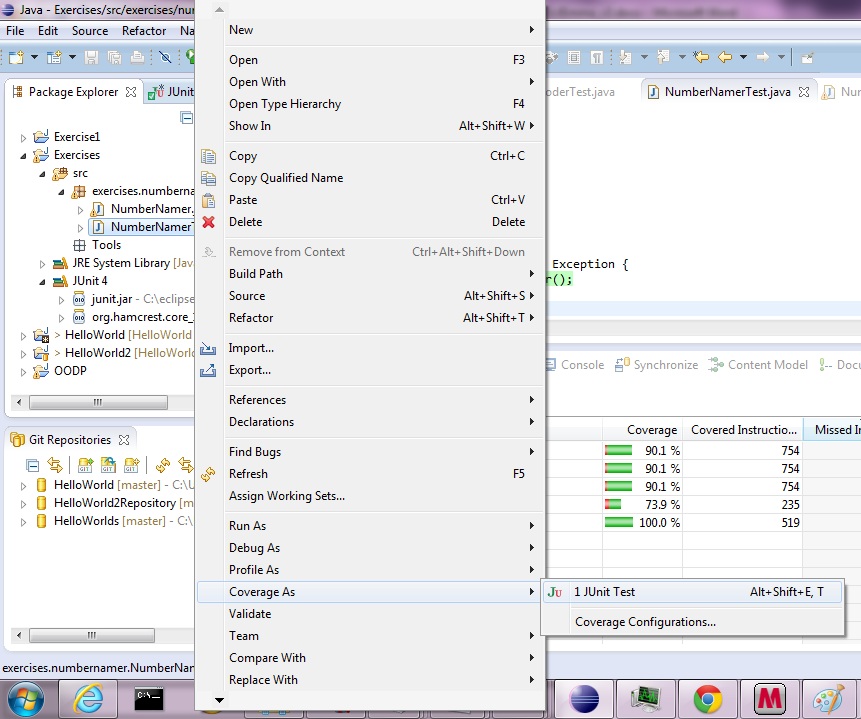
1. Go to the URL above and download the 2 class files to a working directory.

C:\Users\Davison\workspace\exercises\src\exercises\numbernamer is the working directory used in this tutorial

1. Go to a NumberNamerTest in Eclipse Project Explorer and select.

2. Right click for context menu to appear

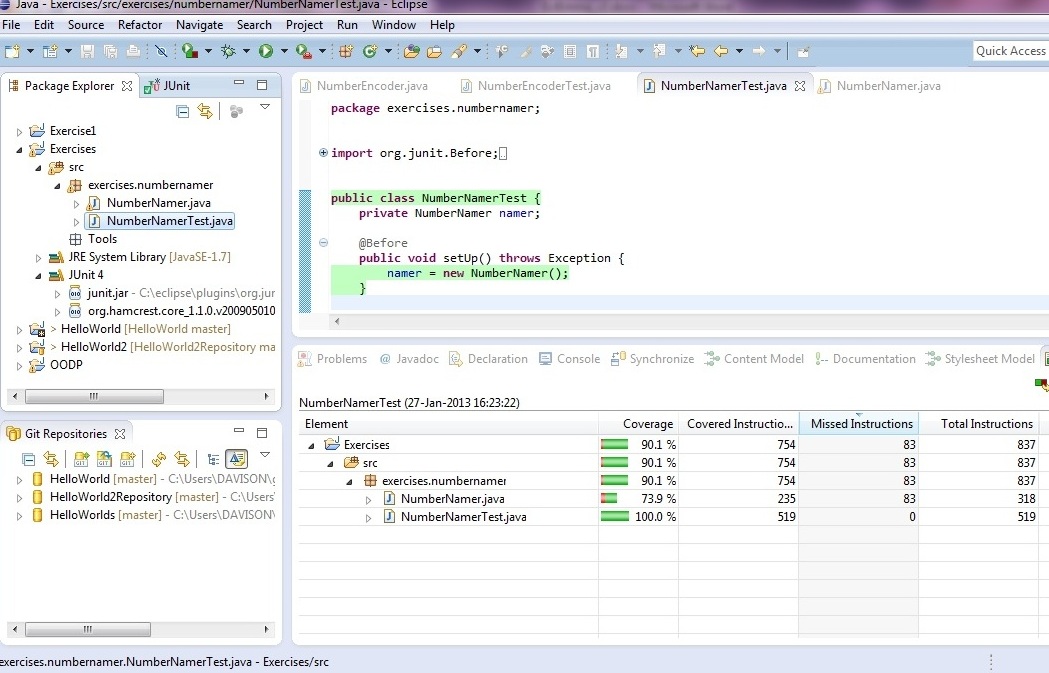
3. Select Coverage As ---> JUnit test from the context menu



The test case/suite will execute and a coverage report will appear in the coverage view dialog (below) and see CodeAfterCoverageLaunch.docx for results of coverage analysis results on NumberNamer.java and NumberNamerTest.java.

CodeAfterCoverageLaunch.docx can be downloaded from the following github repository

<https://github.com/dbullo01/NumberNamer>



**6. A bit about JaCoCo**

**6.1 JaCoCo is used by EclEmma and uses counters to calculate and show the coverage metrics**

**Coverage Counters**

Coverage counters calculate coverage measures.

* Derived from info embedded in Java Class files (java byte code and debug information)
* Allows code coverage when source code does or does not exist.
* Collected info can be mapped back to line level in source code.

**Any known limitations?** - Not all java code constructs can be compiled to byte code. Sometimes unexpected code coverage results are produced as a result. Users should be aware of this

**Types of Counts**

There are counts available for

* Instructions (C0),
* Branches (C1),
* Lines,
* Methods,
* Types
* Cyclomatic complexity.

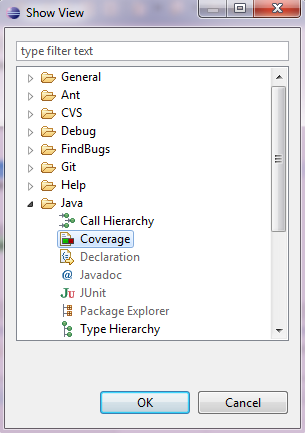
**7. The Coverage View**

The Coverage View is shown when a test/test suite or application is executed. Coverage summaries for the active session are shown automatically after a coverage launch. It can also be opened manually using the following steps:

**7.1 Showing the Coverage Window (Manually)**

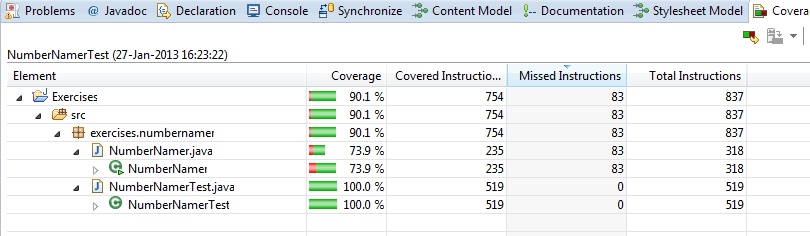
1.Select Window in Eclipse IDE

2.Select Show View and the following dialog will appear



3.Select Java and then Coverage. Press OK button.

The coverage view dialog shows the code coverage results for coverage that was launched by the launch tool (to the left of dialog by default).



The items on the dialog are

* Coverage ratio (% of item covered)
* Covered items (count of items covered),
* Missed items (count of items missed or not covered)
* Total items (total no of items)

Where item is a type of count listed in 6.1.

The items in the project and shown in the left of the Coverage Window are known as elements of Java elements.

The summary stats above are produced for the child elements listed for each of the analyzed parent java elements within the Java hierarchy (such as source code files). Coverage results are on the right of the Coverage Window.

**7.2 Selecting to View Source Code that is related to the code analysis results**

1. Select Window menu in Eclipse, then Show View and Select Other menu items.

2. A Show View dialog will appear. Select Coverage and the Coverage View Window will appear

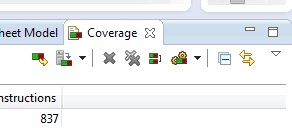
3. Double clicking on an element name (e.g. NumberNamer.java) in the Coverage View Window

4. The source code for the java element should appear in the Eclipse IDE code window with the code highlighting (annotation) as result of code coverage analysis.

5. Optionally click on the element name header to sort java elements in descending or ascending order.

**The Coverage View Toolbar**

There is also a toolbar for the coverage view in the Coverage View Window

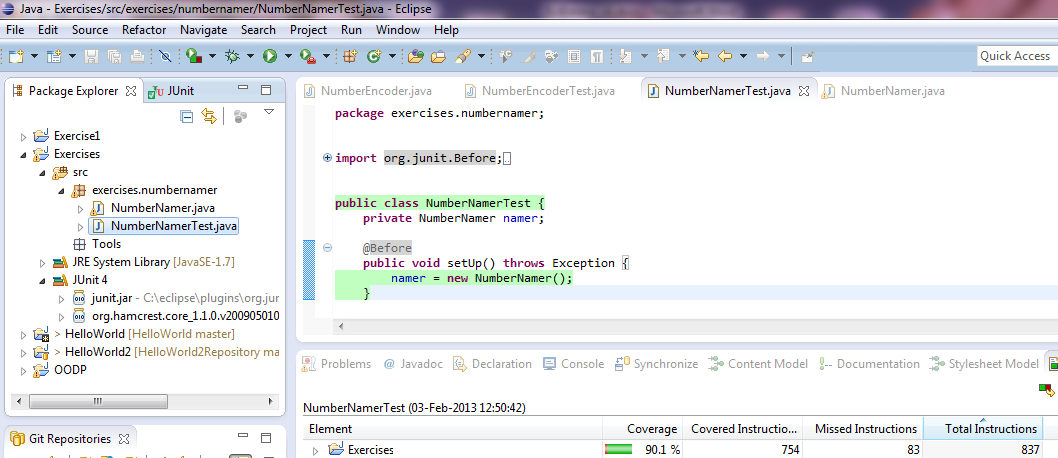


This tool bar allows the following functionality:

** Coverage last launched** – The user can choose to re-run coverage session that is currently selected i.e. for example the NumberNamerTest JUnit launch coverage (Screenshot below).

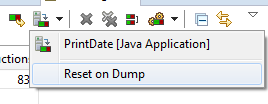
1. Selecting the Java element in the Eclipse Package Explorer Window.

2. Selecting the Relaunch Coverage Session button. The application or Unit test (in this case) will run and produce Coverage session including coverage data and highlighting (annotation) of source code.



**Dump Execution data** – Allows a new session to be created from the data that is dumped from a running process. One process must be running in Coverage Mode for this to functionality to be available

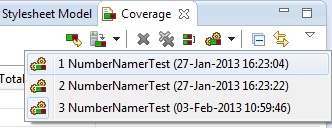
Below shows an application process (PrintDate class) that is running. Selecting this process allows users to dump execution data to create a new coverage session.



**Remove active session** – Allows the coverage session that is selected to be removed.

**Remove all sessions** – Allows all coverage sessions to be removed

** Select active session** – Allows a session to be made the active one by selecting from a drop down menu.

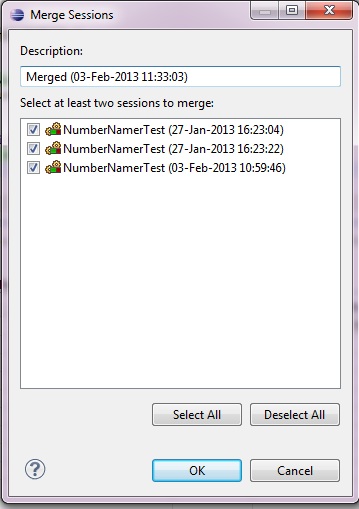


**Collapse all** – All tree nodes in the Coverage View window will be expanded when the user selects this.

** Link with current selection** – If java element is selected in the coverage view with this option toggled on then the same element in other views or the Eclipse editor is shown.

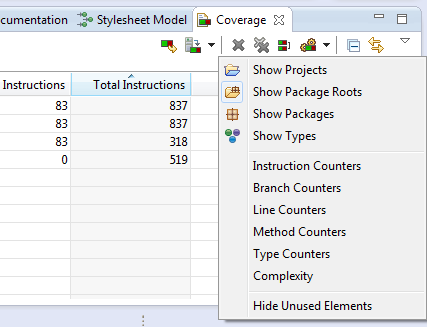
**Merge Sessions** – More than one session can be merged into one by selecting the more than 1 session from those available in the dialog to merge

The following dialog is displayed when the user clicks this button

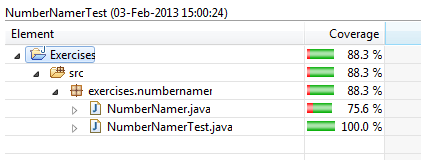


NOTE: If there is no session or there is a single session then some of the options above are deactivated. More settings on the coverage view drop down menu is available (see below)

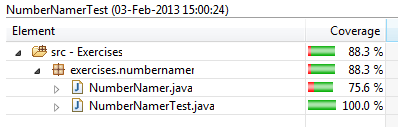
Coverage View Drop Down menu (When clicked menu appears)



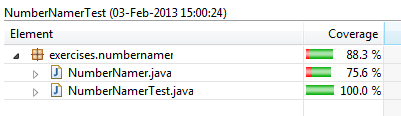
**Show Projects -** Selecting this shows projects in Coverage View

****

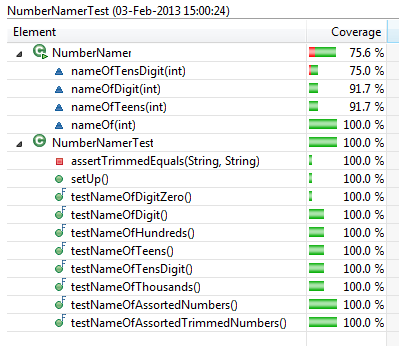
**Show Package Roots** - Selecting this shows the package root



**Show Packages -** Selecting this shows all the package(s) for the coverage

****

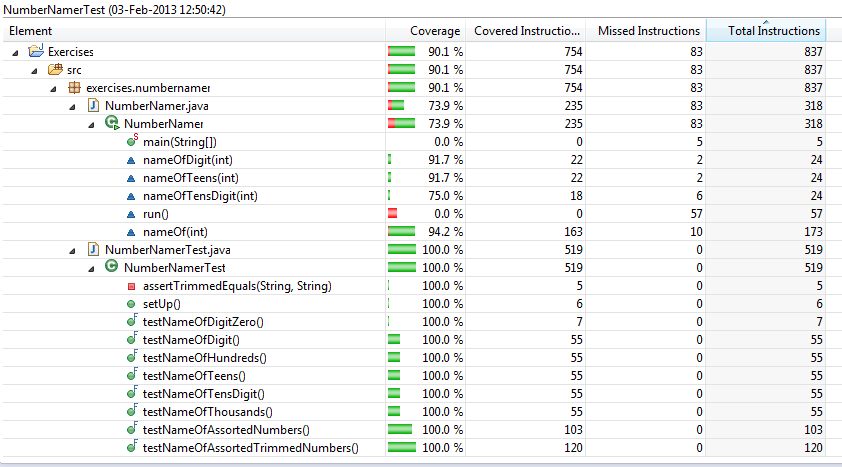
**Show Types -** Selecting this option shows all the types used in the code and analysed in the code coverage analysis

****

**Instruction Counters (Please refer to CodeAfterCoverageLaunch.docx to compare to Source Code)**

1. Select this option from the Coverage Drop down menu changes the Coverage View window to show all the counts relating to the Instructions in the code.

2. The default Coverage View counters will be Instruction counters but will change to these if they are not already of this type.

****

Below is a code snippet showing a function from NumberNamer java class file

String nameOfTensDigit(**int** digit) {

**switch** (digit) {

**case** 0:

**return** "";

**case** 1:

**return** "ten";

**case** 2:

**return** "twenty";

**case** 3:

**return** "thirty";

**case** 4:

**return** "forty";

**case** 5:

**return** "fifty";

**case** 6:

**return** "sixty";

**case** 7:

**return** "seventy";

**case** 8:

**return** "eighty";

**case** 9:

**return** "ninety";

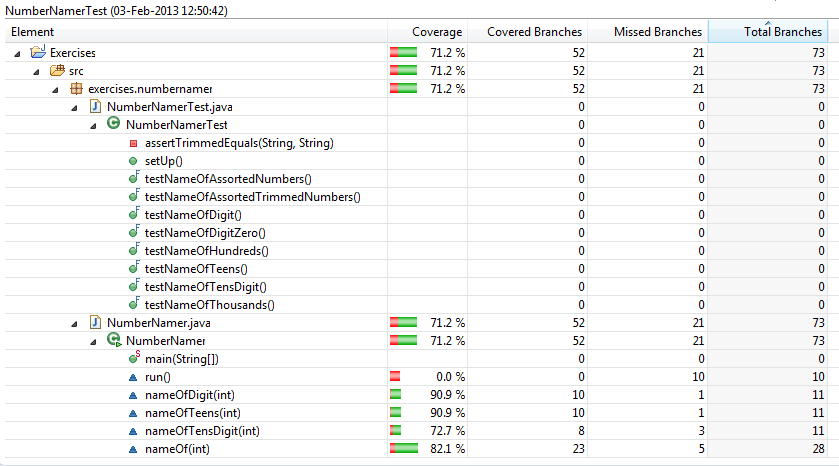
}

**return** "???";

}

**Branch Counters (Please refer to CodeAfterCoverageLaunch.docx to compare to Source Code)**

Select this option to show counts for Branches in the code covered/not covered by unit tests after code coverage analysis has been run using EclEmma.

****

String nameOfTensDigit(**int** digit) {

**switch** (digit) {

**case** 0:

**return** "";

**case** 1:

**return** "ten";

**case** 2:

**return** "twenty";

**case** 3:

**return** "thirty";

**case** 4:

**return** "forty";

**case** 5:

**return** "fifty";

**case** 6:

**return** "sixty";

**case** 7:

**return** "seventy";

**case** 8:

**return** "eighty";

**case** 9:

**return** "ninety";

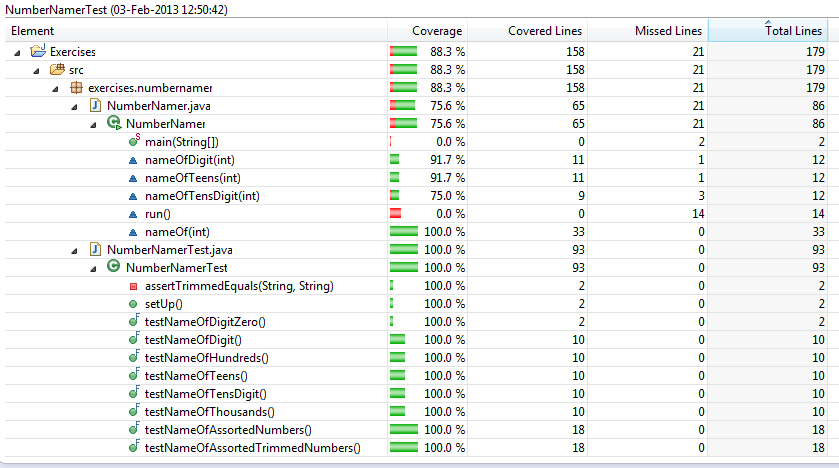
}

**return** "???";

}

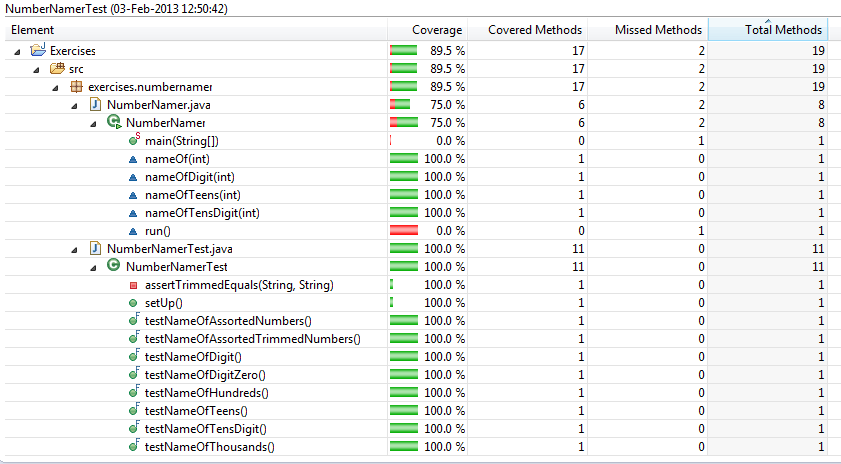
**Line Counters (Please refer to CodeAfterCoverageLaunch.docx to compare to Source Code)**

Select this option to show line counts for the lines of code covered/not covered by unit tests after running EclEmma.

****

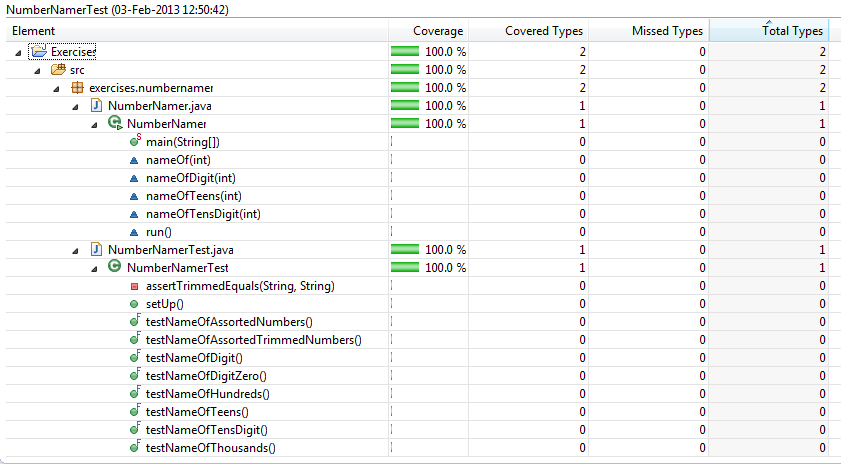
**Method Counters (Please refer to CodeAfterCoverageLaunch.docx to compare to Source Code)**

Select this option to show method counts for the methods in the source code covered/not covered by unit tests after running EclEmma for code coverage analysis.

****

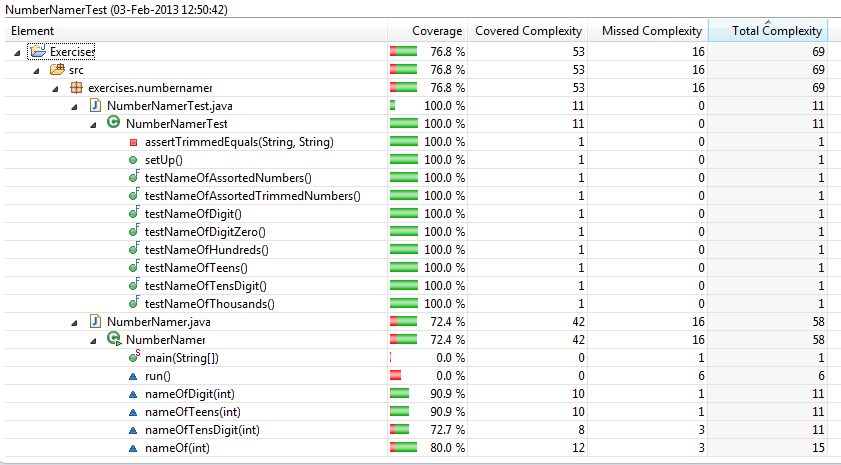
**Type Counters (Please refer to CodeAfterCoverageLaunch.docx to compare to Source Code)**

Select this option to show type counts for the types in the source code covered/not covered by unit tests after running EclEmma for code coverage analysis.

****

**Complexity Counters**

Select this option to show Cyclomatic Complexity counts for the Cyclomatic Complexity in the source code covered/not covered by unit tests after running EclEmma for code coverage analysis.

****

**Hide Unused Elements** - Select when you would like all elements that have not been executed during the coverage session can be filtered from the coverage view.

**8. Annotation of Source Code**

In the IDE code window the source code and any source code in external libraries is highlighted with different colours.

Each colour indicates what code has been executed from those lines that have not been executed or partially executed.

 Fully covered lines

 Partly covered lines - (some branches or instructions were missed).

 Not executed lines - Red

**Example**

String nameOfTensDigit(**int** digit) {

**switch** (digit) {

**case** 0:

**return** "";

**case** 1:

**return** "ten";

**case** 2:

**return** "twenty";

**case** 3:

**return** "thirty";

**case** 4:

**return** "forty";

**case** 5:

**return** "fifty";

**case** 6:

**return** "sixty";

**case** 7:

**return** "seventy";

**case** 8:

**return** "eighty";

**case** 9:

**return** "ninety";

}

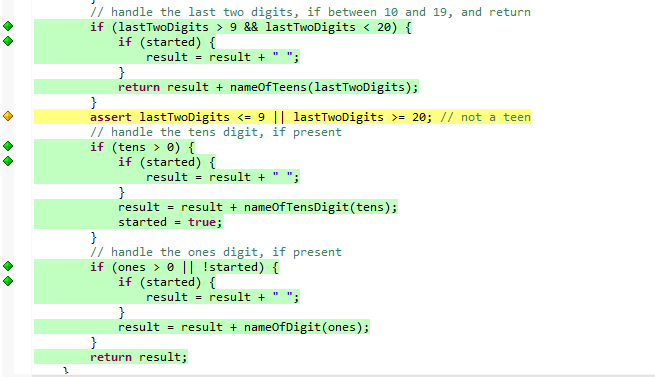
**return** "???";

There are diamond symbols also that appear in the coverage window and are coloured similarly to the covered lines. These symbols represent branches in the code:

 Fully covered branches

 Partly covered branches

 Not executed branches from a line



Above code snippet from NumberNamer,java showing Branch Coverage (see CodeAfterCoverageLaunch.docx)

Colours for the covered lines and branches can be set by users in the Coverage Preferences option (see Changing Highlight Colour below).

**Caution:**

Highlighting is based on the underlying JaCoCo library and how byte code is produced to represent lines of code in java class files. The JaCoCo library only works with java class files. Byte code may be produced for no obvious reason to the user affecting the highlighting. The effect of this is that the highlighting might not make sense to the user.

**Changing Highlight Colour**

This can be done by going to Preferences in Eclipse menu.

1. Select "General"

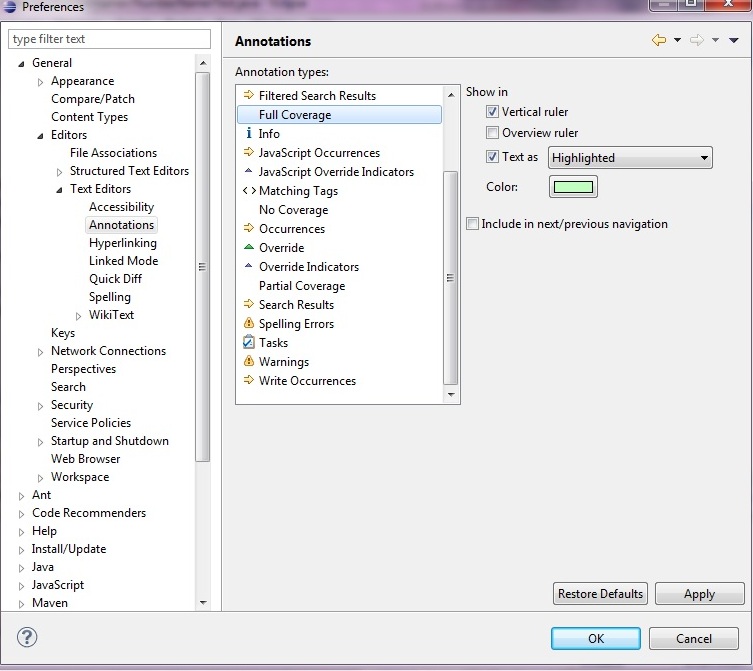
2. Select "Appearance"

3. Select "Editors"

4. Select "Text Editors"

5. Select "Annotations"

The code highlighting colour for each of Full Coverage, Partial Coverage and No Coverage can be changed in Annotation Types list box.



**Note:** When a source file or project is edited the source annotation (highlighting) will disappear.

**9. Coverage Properties**

All the different types of Code Coverage count reports summarizing each Java element (Java project, package, source folder, type or method) can be viewed together and shown all in one report. This is done via the Code Coverage page.

**To view the code Coverage Page**

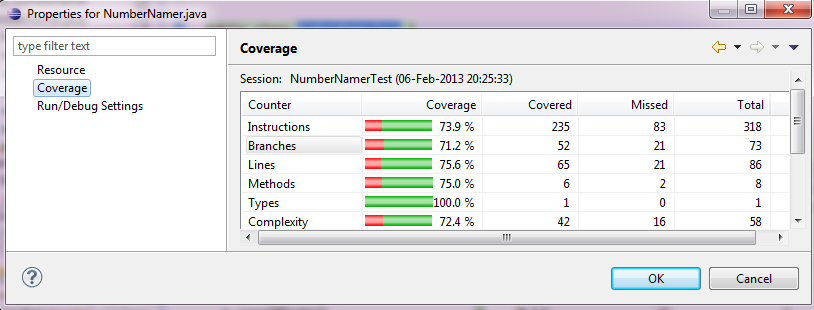
1. Launch a coverage within an active coverage session. (Stats are only available after a coverage has been launched within an active coverage session). Try using NumberNamer or NumberNamerTest to launch as a coverage.

2. Right click and select Properties from context menu of Coverage Window view

OR

2. Right click and select Properties from context menu of Package Explorer

3. Select the coverage item on the left of the Properties dialog. Coverage counts should appear for the Java element being analyzed (NB Counts below are being displayed for NumberNamerTest).

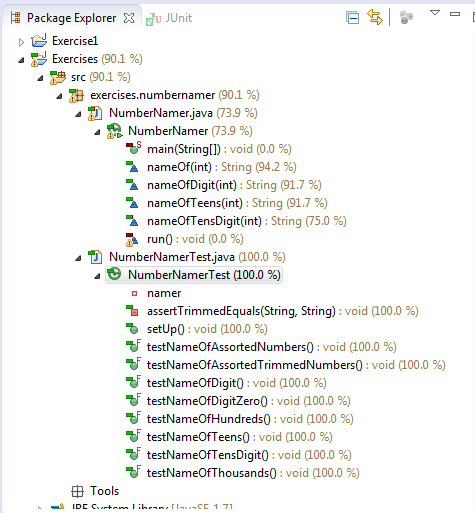
****

**10. Coverage Decorators**

Decorators are an optional feature in EclEmma and are not shown by default but can be turned on to aid code coverage analysis, further code improvement and re-analysis. Decorators are text or graphic which is assigned to Java elements that appear in a coverage session in the Eclipse IDE Package window to allow users to see headline statistics/measures.

They are not shown if a java element does not contain executable source code. An example of where this would be the case is Java Class Abstract methods.

The elements names have a green or red bar assigned to the elements icon (left of element name). On the right of the element name is a percentage score. This percentage represents a score based on the instruction type counter.



NOTE: The decorators are not visible if a coverage session is not active.

**Showing Decorators using Eclipse IDE**

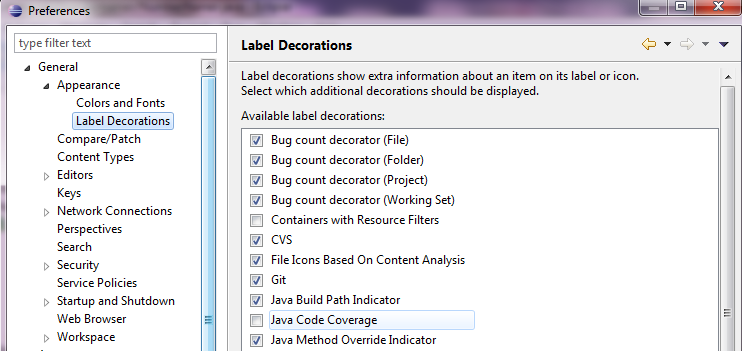
1. Select Window menu in Eclipse and select Preferences menu item

2. The Preferences dialog will appear.

3. Next Select General

4. Select Appearance

5. Select Label Decorators



6. Finally select Java Code Coverage

7. Press OK, the Code Coverage decorators should be turned on.

**11. Coverage Sessions**

Coverage session is the result of a coverage run of a program

There are options in EclEmma to manage Coverage Sessions where more than one may exist:

**11.1. The Lifecycle of a Coverage Session**

* Sessions are created after a launch of a coverage
* Sessions can be deleted using the Coverage View
* Sessions can be imported from external launches
* All Coverage Sessions within EclEmma are removed when the Eclipse IDE workbench is closed.

**11.2 Choosing the Active Session**

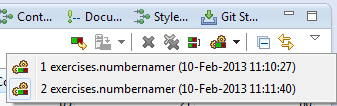
There is only one active coverage session at a time from the multiple launches that are possible

Only one session can be active. Users can choose from multiple sessions that have been created from previous runs/ launches which one is the active session.

Choosing a Coverage Session as the Active Session from a list of multiple sessions

1. Goto Coverage View and Select Drop down list icon. 

2. Select "Active Session" from the drop down list. (Below is an example of 2 sessions. It is the numbernamer package with the java class and java JUNIT test class run twice to create the 2 sessions)



The selection defined the input for the Coverage View dialog and the java source code that will have highlighting.

**11.3 Choosing to Merge Sessions**

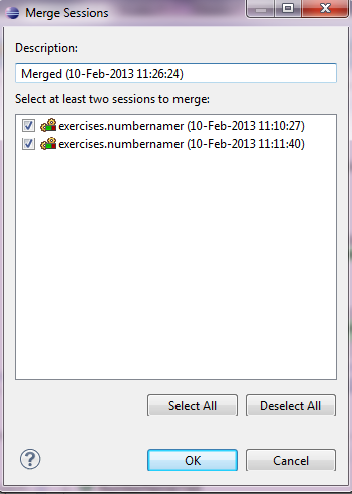
Analysis is easier form a single coverage session. Multiple coverage sessions might exist due to different runs. Multiple sessions can be merged into one single session if it makes sense to do this for the required code coverage analysis.

1. Go to Coverage View window

2. Click the Coverage View window

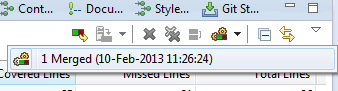
3. Select Merge Sessions icon 

The user can combine a single coverage session with a selection of sessions from those that are available. So choosing the 2 sessions below will be merged into a session called Merger (10-Feb-2013 11:26:24)



4. Click ok.

5. Click Active Sessions icon. The new session will appear as the only session available and thus the only active session in this case. Notice the greying out of other available functionality as a result.

****

**12. Importing and Exporting Sessions**

**12.1 Importing Sessions**

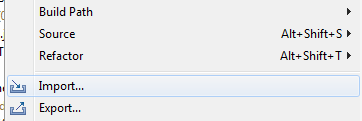
Sessions can be imported from .exec files. This is done for programs / applications that are launched outside of the EclEmma/Eclipse IDE workbench environment. These externally launched applications can produce .exec files via JaCoCo. The .exec files allows source code from these externally run programs to be imported into EclEmma and analyzed.

**Starting the import wizard:**

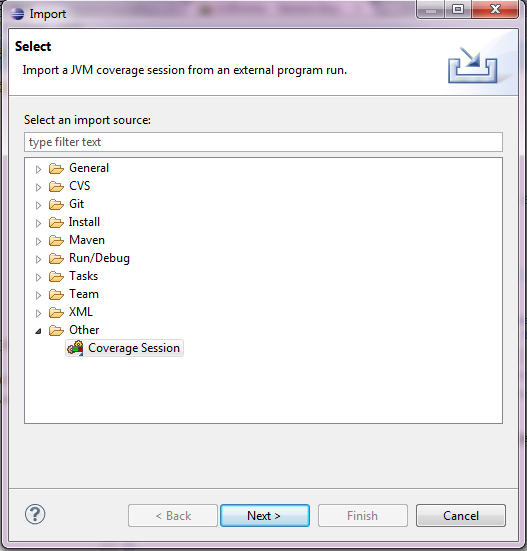
1.Select File in the Eclipse File menu

2.Selecting Import from the File Menu in Eclipse

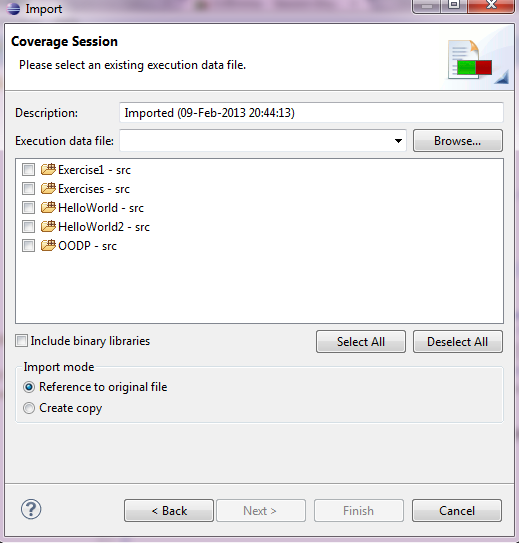
Alternatively the Import Wizard can be accessed from the context menu by right clicking Package Explorer and selecting Import..

****

When Import Wizard appears:



1. Select Other and then Coverage Session. Click Next button



1. Give name to Imported Session in Description

2. Select an existing coverage or session file with.ec or .es file extension in the Execution data files section.

3. Select source folder or libraries.

**NOTE:** The execution data that is imported has to be based on class files used within the Eclipse IDE. If different class files produced using a different IDE were used for the launch then no coverage will be produced.

**12.2 Exporting a Session**

There needs to be at least one coverage session available before the export wizard can be used. This can be checked by Selecting Active Sessions button  from the Coverage View window before proceeding.

**Coverage sessions can be via the Export Wizard into following file formats**

|  |  |
| --- | --- |
| **FILE FORMATS** | |
| **HTML** | A set of HTML files that give reports |
| **ZIPPED HTML** | One file that gives the same content as the format above |
| **CSV** | Class level data saved in a comma separated format that is widely used by many applications as an import export format. |
| **Execution data** | Native format for JaCoCo execution data |
| **XML** |  |

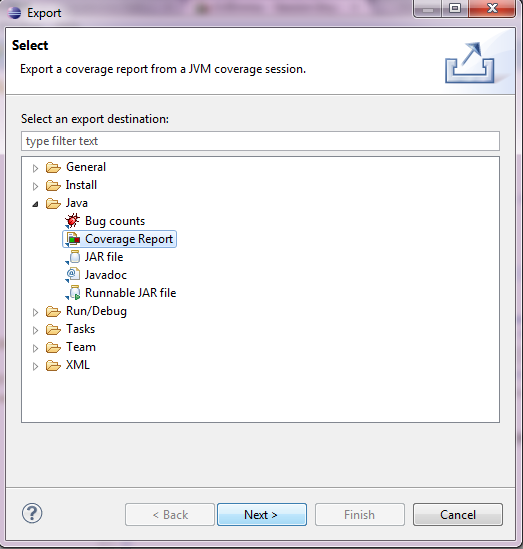
**To access the Export Wizard**

If a coverage session does exist:

1.Select File in the Eclipse IDE File menu

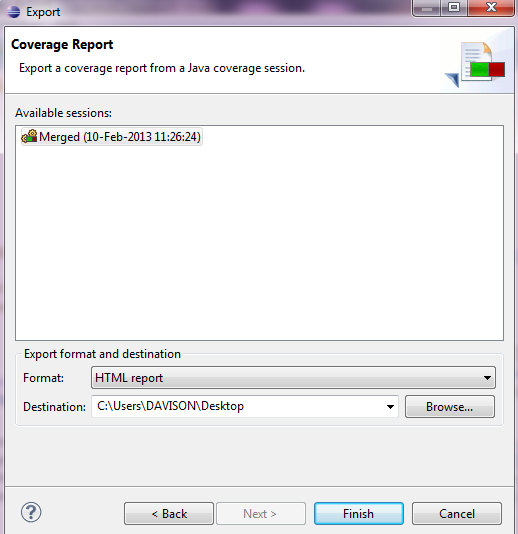
2.Select Export from the File Menu in Eclipse. Alternatively the Export Wizard can be accessed from the context menu in Package Explorer

3. Select Java and the Coverage Report in the Select page of the Export Wizard.



4. Click Next button.

5. Select a coverage session (should be at least one available to choose from)

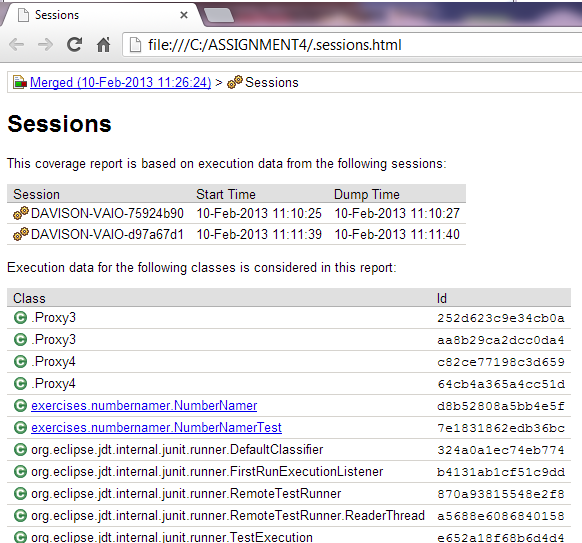


6. Select the export format data type (HTML,HTML packed into zip, XML, JaCoCo execution data, or CSV file). For now select HTML

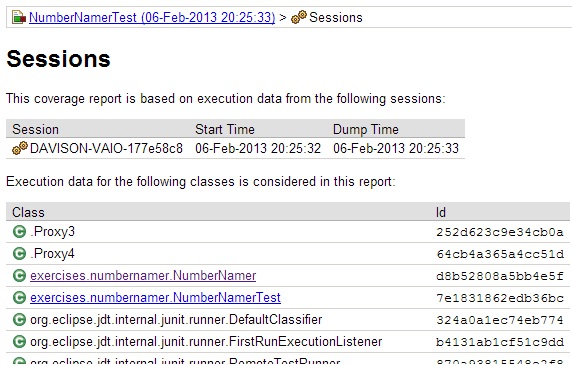
7. Select the output path for the exported coverage session using the browse button.

8. Press Finish.

An extract from a HTML Report with navigable links is below for the Coverage Session above (NB your session and resulting report may be slightly different depending if you are viewing (a) an exported merged session or (b) an exported single session).

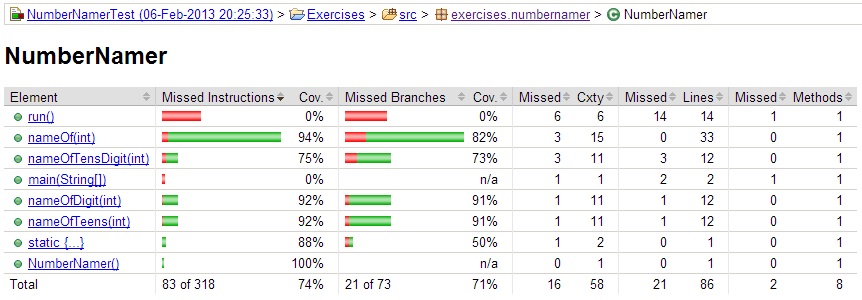


Above is an example of (a) report for a merged session



Above is an example of (b) report for a single session

8. Click NumberNamer or NumberNamerTest to see coverage report for that file.



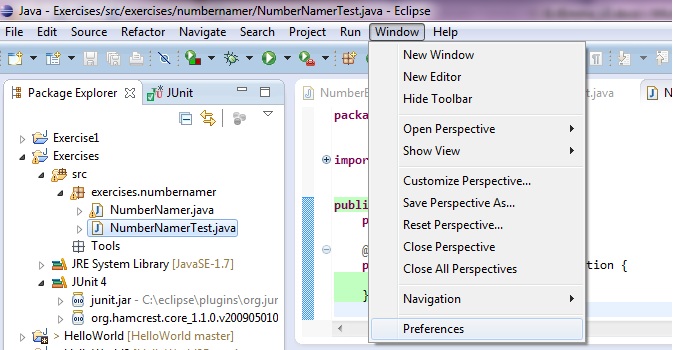
**14. Preferences for Code Coverage**

**14.1 The behaviour of EclEmma can be adjusted using the Preferences dialog.**

1. Select Window in Eclipse IDE

2. Select Preferences,

the Preferences dialog should appear.

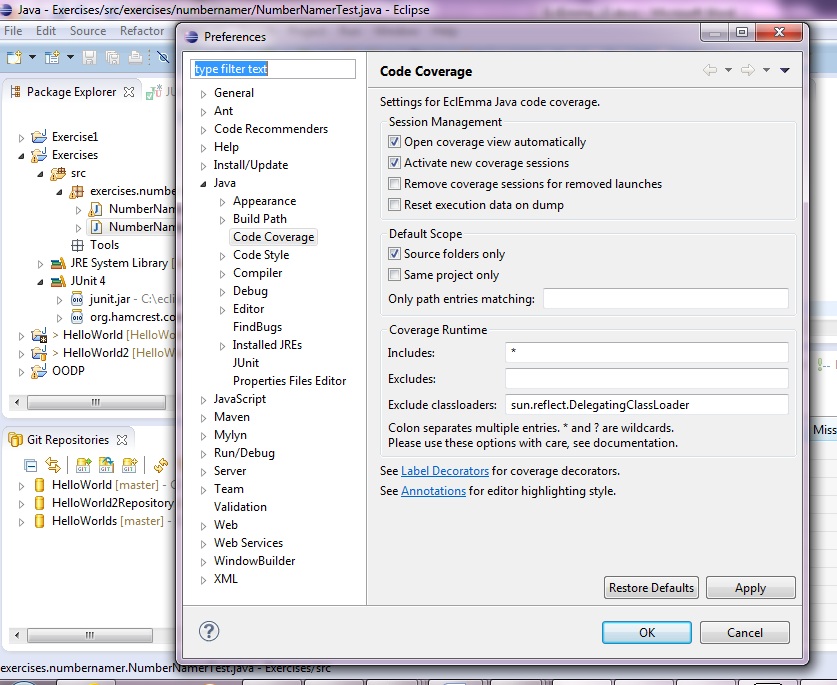


3. Select Java on the left hand side of the preferences dialog from the list of options.

i.e. Sub options will appear as an expanded list.

4. Select Code Coverage from the sub options.

5. The Code Coverage options will appear in the Preferences dialog (below)



The following Code Coverage options are available as user selectable checkboxes on the Preferences dialog for

**Session management:**

**Open coverage view automatically (on by default)** - Select this option to allow the Coverage Window to be opened automatically and shown in the Eclipse Workbench window when a new coverage session is made active.

**Remove coverage sessions for removed launches (Off by default) -** Select this option to allowsessions will be deleted from the Coverage View automatically if this option is turned on. The default is that coverage session(s) stay in the coverage window until they are manually deleted.

**Activate new coverage sessions (On by default) -** Allows a new coverage to be automatically activated when a new session is imported or created. Result data from the import or new session is shown in Eclipse Java Editor and Coverage View Window.

**Resets execution data (off by default) -** Resets execution data on dump from a running application allowing coverage data at any time to be dumped from Coverage View. Users can use this option to reset execution data after an intermediate dump.

Class path entries are selected by EclEmma to be in scope for code coverage analysis. Users can use the following options Default Scope analysis by default from the section of the Preferences dialog to set scope

**Default Scope :**

**Source folders only (Default on):** Select this to use source based class paths only

**Same projects only (Default off):** Use to select class paths that are from the same project. This will only work for launches that have projects associated with them such as those for Java applications.

**Only path entries matching:** This is a list of comma separated list of strings matching class path entries acting like a filter. Setting this option means that a filter must match a class path entry e.g. "src/main/java"

Please Note: It is important that each launch of a coverage has class path entries for the project that match the settings. If this is not the case then it could result in there being empty analysis scope.

**14.2 Including or Excluding classes from Code Coverage Analysis**

Users can choose to include and exclude classes from code coverage analysis if they are not required by checking the relevant option in the Preferences dialog above. These tend to be used only for performance optimization.

**Includes:** This is a list of class names separated by a colon to include in the code coverage analysis on execution.

**Excludes:** This is a list of class names separated by a colon to exclude from the code coverage analysis on execution.