

EECS4313 – JaCoCo Tutorial

Overview

Motivation

Code coverage is a measure (in percent) of the degree to which the source code of a program is executed when a particular test suit is run. A program with high test coverage has more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.¹

Test coverage was among the first methods invented for systematic software testing. While [JaCoCo](#) is one of the most popular free and open-source toolkit for measuring and reporting Java code coverage.

Goal

The purpose of the document, is to introduce you to the JaCoCo tool. Including how to setup, configure and run this tool to view and export the test coverage report of a sample code, based on some Junit tests.

Most of this text has been taken from <https://www.eclEmma.org/index.html> . I encourage you to refer to this URL, for more information.

Getting started

Prerequisites

This document is focusing on using JaCoCo tool with **Eclipse IDE**, make sure you have the latest version of Eclipse installed (at least version 3.8 or higher), as well as Java 1.5 or higher.

Setup

Eclipse IDE is already bundled with [EclEmma](#), a free Java code coverage tool for Eclipse, which is based on the JaCoCo code coverage library.

You can verify if you have EclEmma installed in your Eclipse by:

1. From your Eclipse menu, select **Help -> Eclipse Marketplace...**
2. Type "**EclEmma**" in the search box.
3. If it's installed you can see the status is **`Installed`**, otherwise you should install this tool by following the installation wizard.

¹ Brader, Larry; Hilliker, Howie; Wills, Alan (March 2, 2013). "Chapter 2 Unit Testing: [Testing the Inside](#)". [Testing for Continuous Delivery with Visual Studio 2012](#). Microsoft. p. 30. ISBN 1621140180. Retrieved 16 June 2016.

Starter Code

For this tutorial, you don't need to have the exact code as the one in this documentation. The sample code discussed in this documentation is for illustration purpose, you can open any Java project as long as there's some JUnit test cases, then follow this document.

Guide

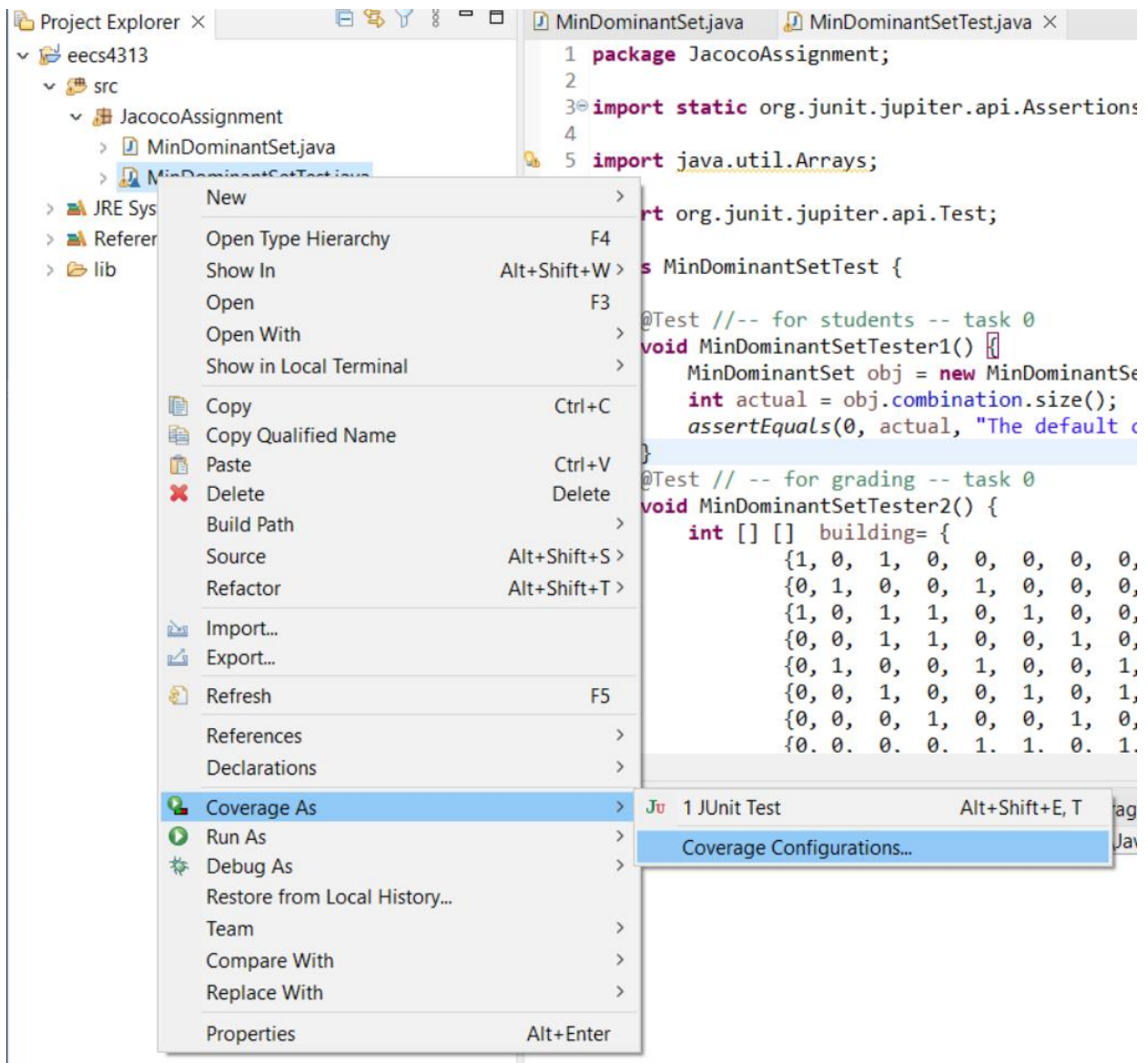
Launching the Coverage Mode

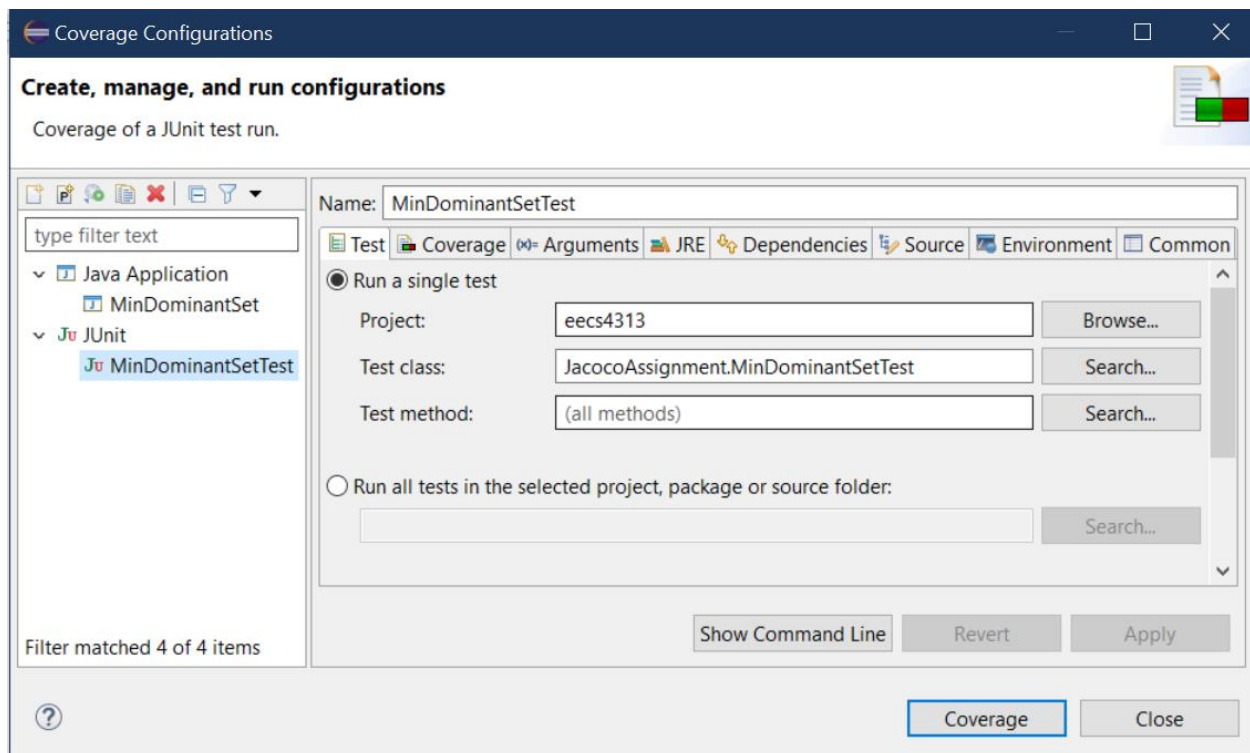
Eclipse allows running Java programs directly from the workbench. Programs can be launched in different so called *launch modes*. Normally you may launch your program in *Run* or *Debug* mode. Now we are going to run the program in **Coverage** mode, which is available from the *Run* menu and the toolbar:



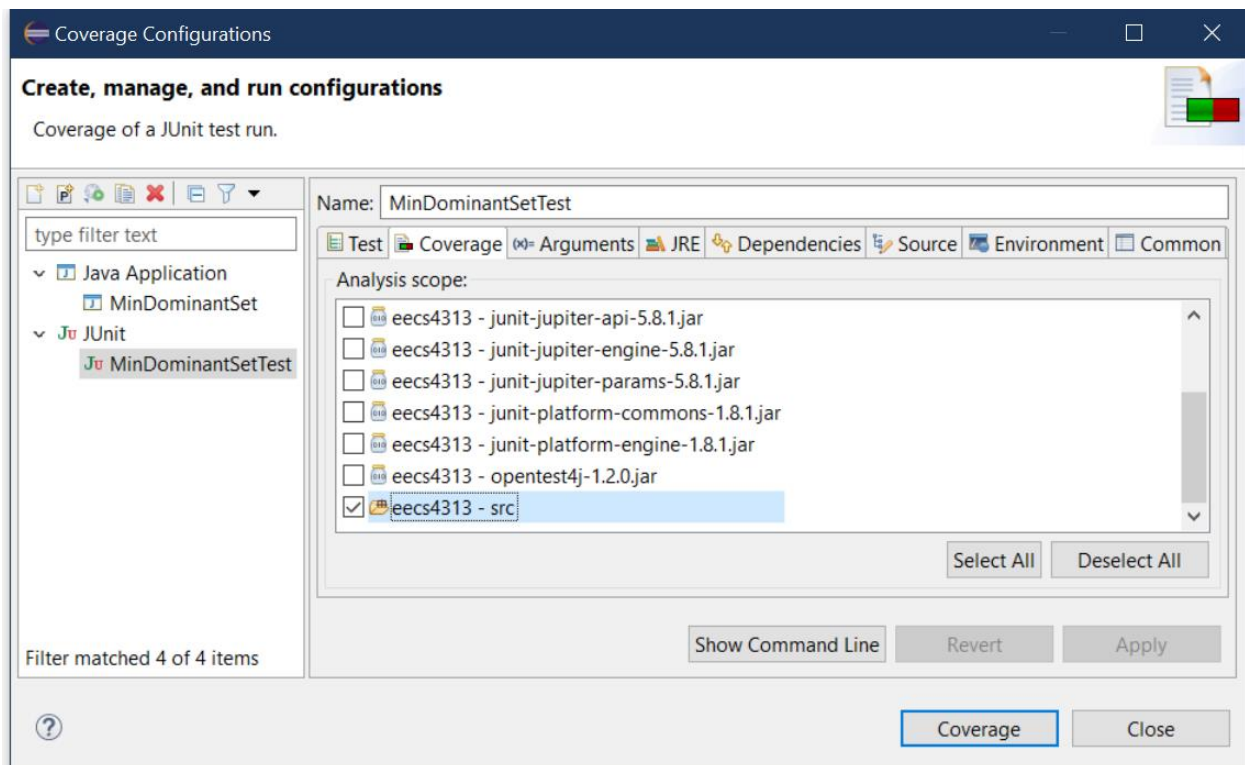
Note: If the Coverage drop-down toolbar button is not visible in your current workbench perspective, open the Customize Perspective... dialog and enable the Coverage command group on the Commands tab.

Existing launch configurations can be launched directly in **Coverage** mode using default settings. As with the *Run* and *Debug* mode you might also select a Java element and launch it directly from the **Coverage As** context menu. If required some settings can be modified in the coverage launch dialog:

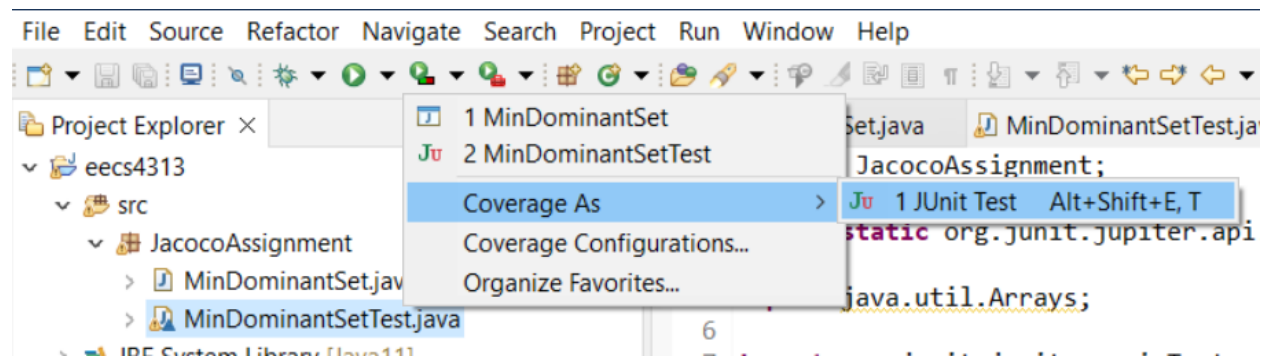




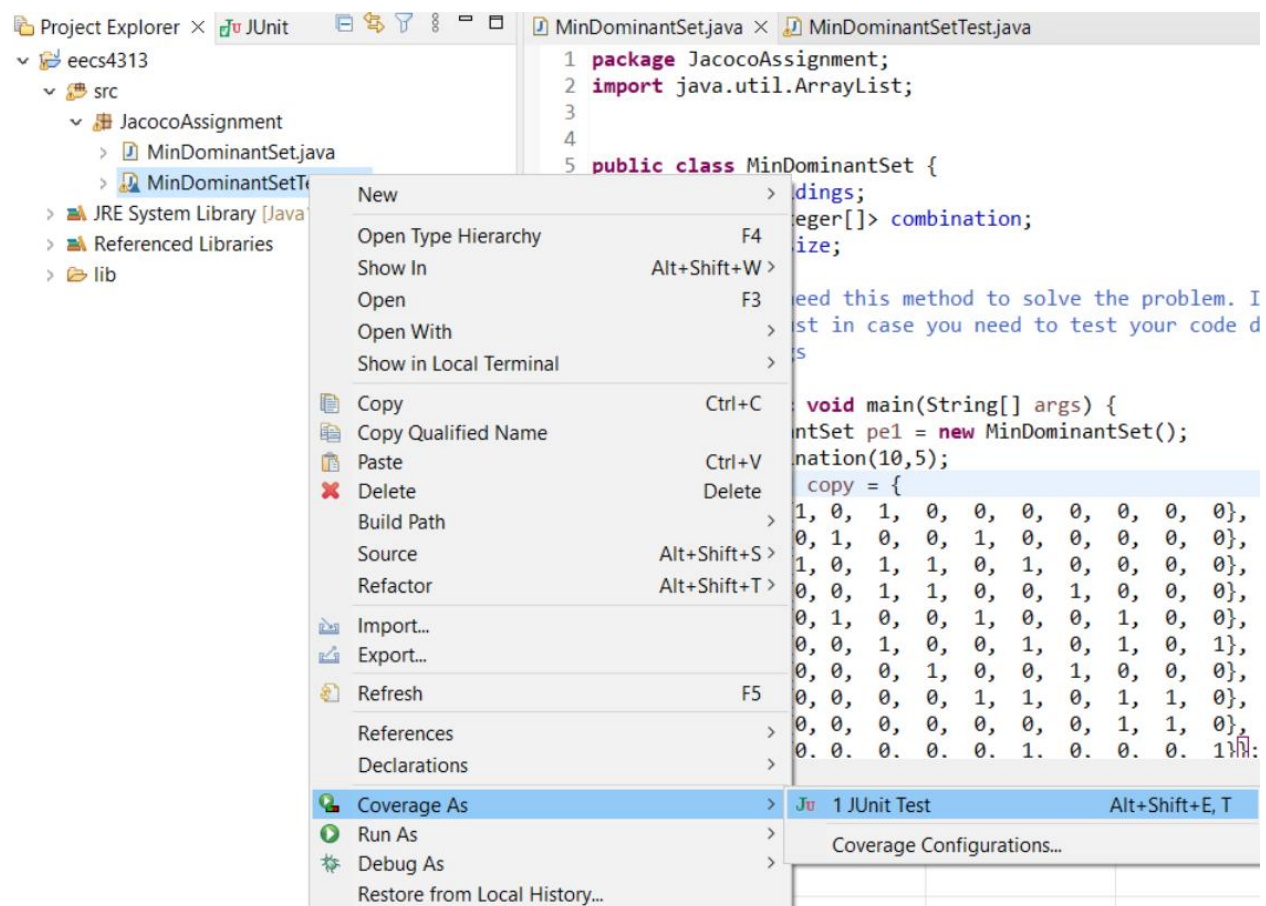
In the **Coverage** tab the Java class path entries for code coverage analysis can be selected. At least one entry must be selected to run an application in **Coverage** mode. The rules which class path entries are selected by default can be adjusted in the code coverage preferences.



After the above setup, you can run the Java program in **Coverage** mode, by:



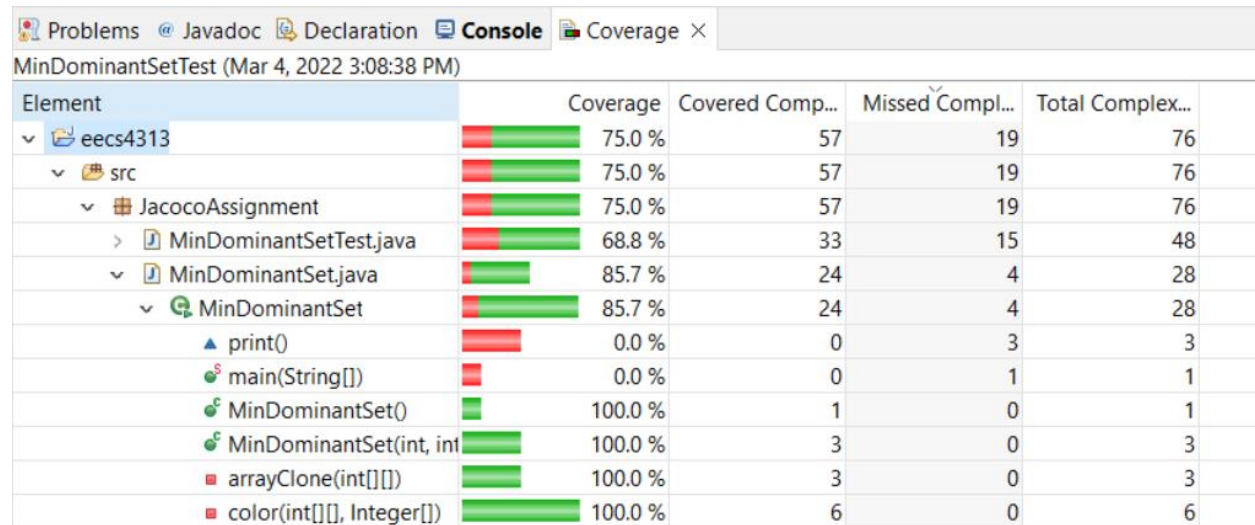
Or through context menu:



This will run the select Junit test class in **Coverage** mode using JaCoCo.

Using the Coverage View

The **Coverage** view will automatically appear after program finishes running in **Coverage** mode, or one can manually opened from the Window → Show View menu in the Java category. It shows coverage summaries for the active session.



MinDominantSetTest (Mar 4, 2022 3:08:38 PM)

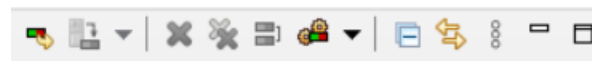
Element	Coverage	Covered Comp...	Missed Compl...	Total Complex...
▼ eecs4313		57	19	76
▼ src		57	19	76
▼ JacocoAssignment		57	19	76
> MinDominantSetTest.java		33	15	48
▼ MinDominantSet.java		24	4	28
▼ MinDominantSet		24	4	28
▲ print()		0	3	3
● main(String[])		0	1	1
● MinDominantSet()		1	0	1
● MinDominantSet(int, int)		3	0	3
■ arrayClone(int[][])		3	0	3
■ color(int[][], Integer[])		6	0	6

The **Coverage** view shows all analyzed Java elements within the common Java hierarchy.

The elements may be sorted in ascending or descending order by clicking the respective column header. Double-clicking an element opens its declaration in an editor with highlighted source code. You can select between different metrics, see last section for details.


Toolbar and Drop-Down Menu

The coverage view's toolbar offers the following actions:



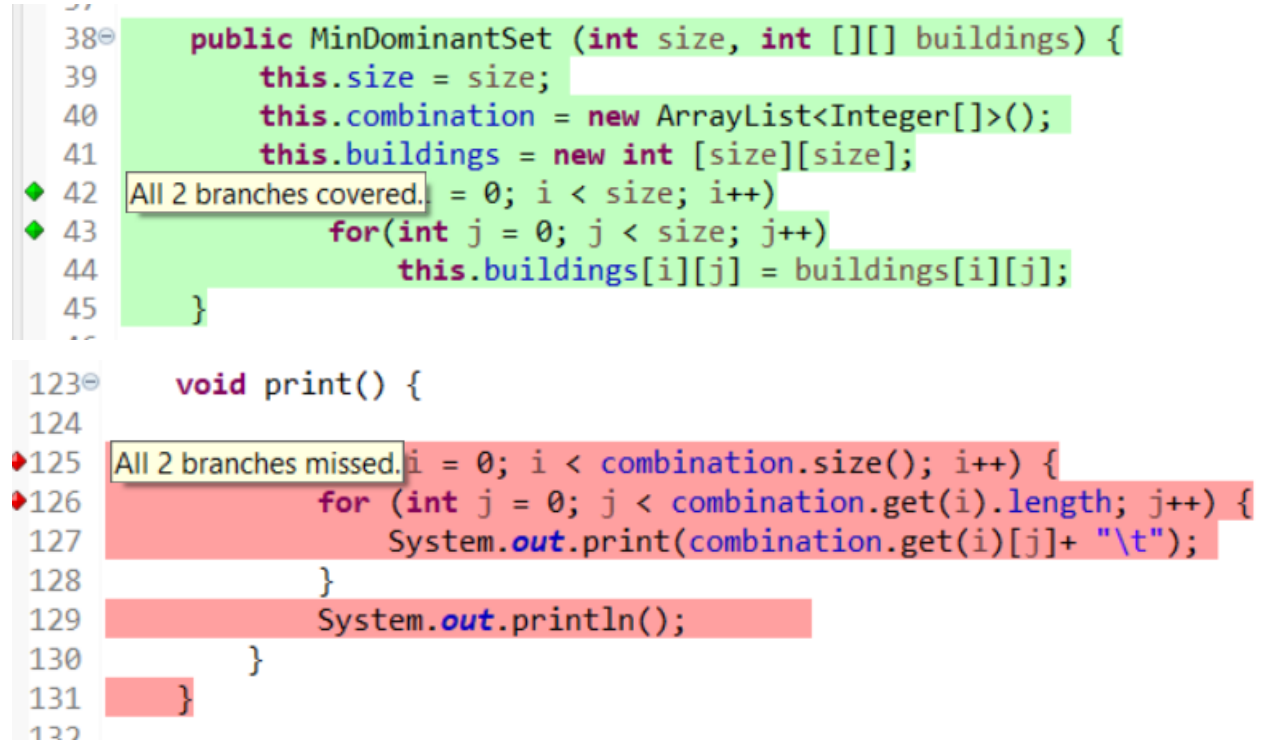
- **Coverage Last Launched:** Re-run the currently selected coverage session.
- **Dump Execution Data:** Dump execution data from a running process and create a new session from the data. Only active when at least one process is running in Coverage mode.
- **Remove Active Session:** Remove the currently selected coverage session.
- **Remove All Sessions:** Remove all coverage sessions.
- **Merge Sessions:** Merges multiple sessions into a single one.
- **Select Session:** Select session from the drop down-menu and make it the active session.
- **Collapse All:** Collapse all expanded tree nodes.
- **Link with Current Selection:** If this toggle is checked the coverage view automatically reveals the Java element currently selected in other views or editors.

Note: click the **Remove All Sessions**  option will clear all the coloring on the source code.

Some of the actions are deactivated if there is no session or only a single session. More settings are available from the coverage view's drop-down menu .

Source Code Annotation

Line coverage and branch coverage of the active coverage session is also directly displayed in the Java source editors. This works for Java source files contained in the project as well as source code attached to binary libraries.



```
38 public MinDominantSet (int size, int [][] buildings) {
39     this.size = size;
40     this.combination = new ArrayList<Integer[]>();
41     this.buildings = new int [size][size];
42     All 2 branches covered. = 0; i < size; i++)
43     for(int j = 0; j < size; j++)
44         this.buildings[i][j] = buildings[i][j];
45 }

123 void print() {
124
125 All 2 branches missed. i = 0; i < combination.size(); i++) {
126     for (int j = 0; j < combination.get(i).length; j++) {
127         System.out.print(combination.get(i)[j]+ "\t");
128     }
129     System.out.println();
130 }
131 }
132 }
```

Source lines containing executable code get the following color code:

- **green** for fully covered lines,
- **yellow** for partly covered lines (some instructions or branches missed) and
- **red** for lines that have not been executed at all.

In addition colored diamonds are shown at the left for lines containing decision branches. The colors for the diamonds have a similar semantic than the line highlighting colors:

- **green** for fully covered branches,
- **yellow** for partly covered branches and
- **red** when no branches in the particular line have been executed.

These default colors can be modified in the Preferences dialog (see next section). The source annotations automatically disappear when you start editing a source file or delete the coverage session.

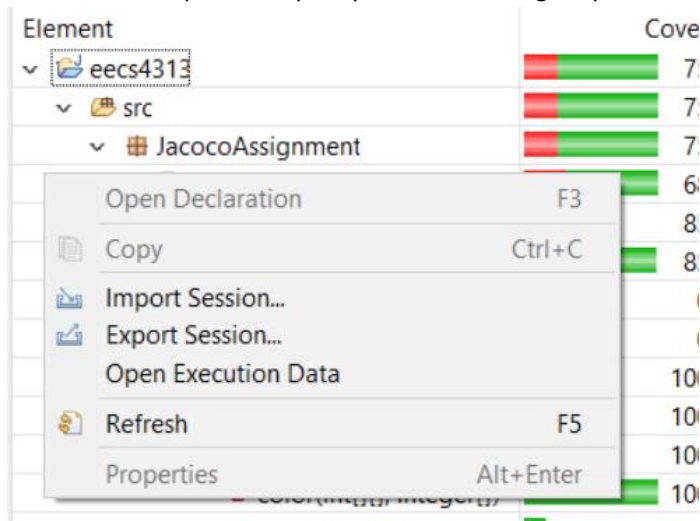
Highlighting Preferences

The Eclipse preferences section **General** → **Appearance** → **Editors** → **Text Editors** → **Annotations** allows to modify the visual representation of coverage highlighting. The corresponding entries are:

- Full Coverage
- Partial Coverage
- No Coverage

Test Coverage Import and Export

You can also import or export your test coverage report:



Session Import

If your program is launched outside the Eclipse debugging environment, you might import JaCoCo execution data from these launches. This allows to study the coverage results directly in your source code. The Coverage Session import wizard can be activated from the **File** → **Import Session...** menu or from the **Coverage** view's context menu.

Session Export

The session export wizard allows to export coverage sessions in one of these formats:





- **HTML**: A detailed and browseable report as a set of HTML files.
- **Zippered HTML**: Same as above but zipped into a single file.
- **XML**: Coverage data as a single, structured XML file.
- **CSV**: Coverage data on class level granularity as comma-separated values.
- **Execution data file**: Native JaCoCo execution data format.

The Coverage Session export wizard can be activated from the **File** → **Export Session...** menu or from the **Coverage** view's context menu. There must be at least one coverage session available to use the export wizard.

Select one of the existing sessions and the export format.

As an example, **HTML** export format will produce an **index.html** file, along with other resources/sessions files. Opening the **index.html** will show the coverage report similar to what we saw in Eclipse:

JacocoAssignment

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
MinDominantSet		53%		88%	4 28	22 69	2 10	0 1
MinDominantSetTest		99%		77%	15 48	2 229	0 15	0 1
Total	244 of 2,795	91%	19 of 102	81%	19 76	24 298	2 25	0 2