



# JHawk 5 Documentation– Eclipse plugin manual

Virtual Machinery  
March 2010

Version 1.0

|  |           |
|--|-----------|
| <b>OVERVIEW.....</b>                         | <b>4</b>  |
| <b>NEW FEATURES ADDED SINCE JHAWK 4.....</b> | <b>6</b>  |
| .....  | 6         |
| <b>DOCUMENTATION.....</b>                    | <b>7</b>  |
| <b>INSTALLATION.....</b>                     | <b>8</b>  |
| Known issues.....                            | 8         |
| Analysing Code.....                          | 9         |
| The JHawk Metrics view.....                  | 10        |
| The Dashboard tab.....                       | 10        |
| The System details tab.....                  | 11        |
| The System Packages tab.....                 | 11        |
| The Package Details tab.....                 | 12        |
| The Class Details tab.....                   | 12        |
| <b>THE DASHBOARD TAB.....</b>                | <b>14</b> |
| <b>SYSTEM DETAILS TAB.....</b>               | <b>16</b> |
| <b>THE SYSTEM PACKAGES TAB.....</b>          | <b>17</b> |
| <b>THE PACKAGE DETAILS TAB.....</b>          | <b>18</b> |
| <b>THE CLASS DETAILS TAB.....</b>            | <b>19</b> |
| <b>THE ALL CLASSES TAB.....</b>              | <b>20</b> |
| <b>ALL METHODS TAB.....</b>                  | <b>21</b> |
| <b>THE EXPORT TAB.....</b>                   | <b>22</b> |
| The CSV format tab.....                      | 22        |
| The HTML Tab.....                            | 23        |
| The Create XML Tab.....                      | 24        |
| Creating XML files for the Data Viewer.....  | 24        |

|   |           |
|---|-----------|
| Creating files to load back into JHawk Stand Alone.....   | 25        |
| <b>OUTPUT FORMATS.....</b>                                | <b>26</b> |
| CSV – Standard.....                                       | 26        |
| CSV – Raw.....  | 26        |
| XML – Standard.....                                       | 27        |
| XML – JHawk Interchange Format.....                       | 28        |
| HTML.....   | 29        |
| <b>THE PREFERENCES TAB.....</b>                           | <b>30</b> |
| General Preferences.....                                  | 30        |
| Importing and exporting settings.....                     | 31        |
| Metrics sub-tabs – Edit Metric Column details.....        | 32        |
| The Buttons.....  | 32        |
| Add a metric and edit details of an existing metric ..... | 33        |

# Overview

Since its release in 1999 JHawk has been a leader in the provision of Software Metrics related data to Java developers. Originally released as a stand-alone application it has now evolved to include a command line version and an Eclipse plugin. Functionality has also been added over time to allow the export of the metrics gathered by JHawk in CSV, HTML and XML format.

JHawk has proved itself in many different areas and its customers have reflected that – from Fortune 500 companies to Academic institutions, from Banking to Telecommunications companies and across the globe from Norway to Brazil and from the US to China.

The first version of JHawk was also the first metrics tool to be integrated with an IDE – Visual Age for Java. We continued that tradition by integrating later versions with Eclipse and JHawk 5 is also integrated with Eclipse 3.3 and above.

A consistent feature of JHawk from the very beginning has been the provision of a simple mechanism to allow users to extend JHawk and to make their own metrics. This feature was one of the main reasons for JHawk's success in Academic Institutions. We simplified this in version 4 and in version 5 we have gone a step further, achieving what we thought would be impossible – maintaining the simplicity but increasing the power.

We have also realised that commercial users need a quick overview that gives them an instant summary of the current state of their code. To achieve this we have added a dashboard tab which gives a quick overview of the metrics at System, Package and Class level. The metrics displayed on the dashboard are configurable – whether they are metrics provided by Virtual Machinery or by users themselves.

We have also added a new product to our line – the JHawk Dataviewer – this allows a user to view changes in core metrics over time – for example over a project lifecycle. This is done using the XML files exported by JHawk. This is an extremely powerful tool which is unique in the Java Software metrics area.

This document is designed to satisfy the needs of users of the JHawk5 plugin for Eclipse – whether they are users of previous versions of the product or completely new to JHawk. It provides information on the use, design and modification of the JHawk product.

If you have used JHawk before then you should find the product as easy to use as before and the new features should become obvious as you use the product.

We have identified three categories of user for this new version of the product.

1. Users completely new to JHawk.
2. Users upgrading from JHawk4 to JHawk5 who generally use the product 'as is'
3. Users upgrading from JHawk4 to JHawk5 who have always configured the product to meet their own needs.

For each of these categories we have suggested a subset of this document that you should read to get you up and running as quickly as possible. If you have any difficulties you should consult the FAQs.

1. Users completely new to JHawk.
  - Read the 'Quick start' section.
  - Run JHawk on some of your code
  - Read the 'JHawk5UsingMetrics' document
  - Read the Section in this document on how to set preferences so that you can tune the settings in JHawk to suit your purpose
  - Read other sections of the document as you need them
2. Users upgrading from JHawk4 to JHawk5 who generally use the product 'as is'
  - Read the 'New Features added since JHawk 4' section to see the new and updated features in JHawk5.
  - Read other sections of the document as you need them
3. Users upgrading from JHawk4 to JHawk5 who have always configured the product to meet their own needs.
  - Read the 'New Features added since JHawk 4' section to see the new and updated features in JHawk5.
  - Read the 'JHawk5CreateMetric' document to find out how to modify JHawk without access to the source code.
  - Read other sections of the document as you need them

# New Features added since JHawk 4

We have added a considerable number of new features to the Eclipse plugin since JHawk version 4 -

- **Dashboard Tab** – The dashboard tab allows users to have a quick visual overview of the state of their code. The Dashboard can be configured using the ‘Preferences’ interface for JHawk. See ‘The Preferences Tab’ section in this document.
- **Ability to create new metrics** – New metrics can be added based on existing metrics, data collected by JHawk or data from external sources. See the separate document ‘JHawk5CreateMetric’
- **Click on a class or method to open up its source in an editor** - Double clicking on a table entry in JHawk for either a Class or a Method will open the appropriate source file in an Eclipse editor.
- **JHawk Metrics Interchange format** - Introduction of a metrics interchange format which allows metrics relating to a particular code release being saved for later use in either the standalone product or in the JHawk Dataviewer.
- **Improved HTML export** - HTML export now has indicators which show which metrics in which code artefacts may be causing issues. These warnings are based on the warning levels set in the JHawk Preferences. See the relevant parts of the ‘Export Tab ‘ and ‘Output Formats’ sections.
- **Addition of a ‘raw’ format for CSV export** which makes it easier to manipulate. The previous formats were difficult to use in calculations as there were sub-total headers which had to be removed. See the ‘Export Tab’ section.
- **Improved documentation.** We have moved away from the HTML documentation to a higher quality of PDF documentation.

# Documentation

The following documents are provided with the distribution (depending on which license you have purchased). They are in PDF format and are located in the 'docs' directory of the distribution –

| Document  | Personal | Professional | Demo |
|---|----------|--------------|------|
| JHawk5CommandLineManual – Documentation for the Command line version of JHawk   | No       | Yes          | Yes  |
| JHawk5CreateMetric – Documentation explaining how to create new metrics that can be added to JHawk  | Yes      | Yes          | No   |
| JHawk5DataViewerManual – Documentation for JHawks DataViewer product  | No       | Yes          | Yes  |
| JHawk5EclipseManual – Documentation for the JHawk Eclipse Plugin  | Yes      | Yes          | Yes  |
| JHawk5Licensing – Licensing details for JHawk products  | Yes      | Yes          | Yes  |
| JHawk5UserManual – Documentation for the JHawk stand alone application  | Yes      | Yes          | Yes  |
| JHawk5UsingMetrics – Documentation outlining how to get the best from JHawk and a list of the metrics implemented by JHawk with details of their calculation.. It also includes an introduction to the area of Java code metrics. | Yes      | Yes          | No   |

# Installation

After you have unzipped the JHawk Distribution into a directory you will find a sub-folder named 'plugins'. Close down your Eclipse application if it is already running. Copy the contents of the plugins directory (a single folder with a name starting with JHawk5Plugin) into the plugins directory of your eclipse folder. When you restart Eclipse JHawk will be ready for use.

The Eclipse plugin can be used in Eclipse 3.4 (Ganymede) and above.

As part of the distribution you will notice the following properties files –

- jhawkbase.properties
- jhawkfull.properties
- jhawkbaseplustom.properties

jhawkbase.properties is a version of the properties file with the base level set of metrics that are initially enabled for JHawk. This is the same as the jhawk.properties file that comes with the JHawk Eclipse plugin distribution.

jhawkfull.properties is a version of the properties file with the full set of metrics available to JHawk enabled.

jhawkbaseplustom.properties is a version of the properties file with the base level set of metrics that are initially enabled for JHawk plus the sample metric found in the CustomMetrics.jar that comes with the distribution. For more details about this metric see the 'JHawk5CreateMetric' document.

You can use any of these sets of metrics by copying them to the jhawk.properties file and starting as normal or by using the -p flag and the property file name (JhawkCommandLine only).

Properties loaded from these files can be amended in the usual way – see 'Preferences' section in the documentation.

## **Known issues**

There is an issue with attempting to use Custom Metrics (i.e. metrics that you have written yourself and add to the CustomMetrics.jar) with the Eclipse 3.4 (Ganymede) distribution for the Mac OSX. This issue is not present if the plugin is installed with the Eclipse 3.5 (Galileo) distribution for Mac OSX.



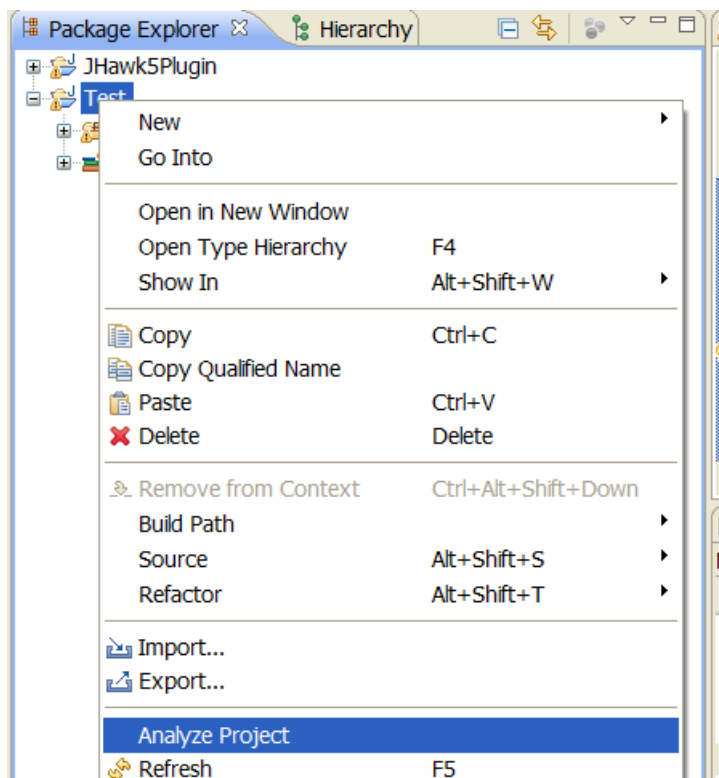
# JHawk Eclipse Plugin – A quick Start

You won't notice any immediate obvious difference when you restart Eclipse after installing the JHawk Plugin. The significant differences are –

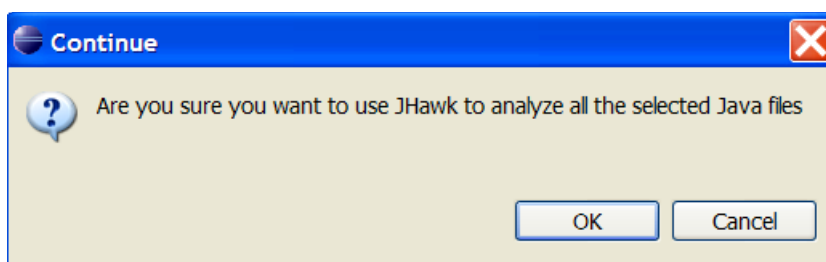
- You will notice an additional option on the right click menu when you select a code artefact (Project, Package, Class etc) in the Project Explorer menu.
- When you go in to the preferences option on the Windows menu you will note an option to change the JHawk preferences. Don't worry about these for the moment – the defaults are adequate to get you started with JHawk.

## Analysing Code

To Analyze code with JHawk simply go to the Project explorer window and select the code artefacts that you wish to analyze (Projects, Packages, Classes). You can multiply select items if you wish. Right click on the selected items and look for the Analyze XXXX menu option where XXXX can be Project, Package or File.



Click on the Menu entry and you will then be asked if you want to carry on with the analysis of the files selected –



If you elect to continue by pressing OK then the busy cursor will appear and JHawk will start analysing your files. The amount of time this will take will depend on the power of your system and the number of files that you have selected.

If you have selected a very large number of files it is important to ensure that you have allocated enough memory to your Eclipse instance to ensure that all the data collected by JHawk can be stored. The heap memory and stack memory allocation in Eclipse is initiated in the Eclipse.ini file. The standard settings are 256M for heap memory and 512k for stack size. The JHawk parser is recursive and can therefore run out of stack space quite easily causing a stack overflow error. We recommend that you allocate 512M of heap memory and 2M of stack memory for general purposes. If you are analyzing very large numbers of files you may have to increase the heap size even more. In practice there is unlikely to be any requirement for the stack size to be above 4M.

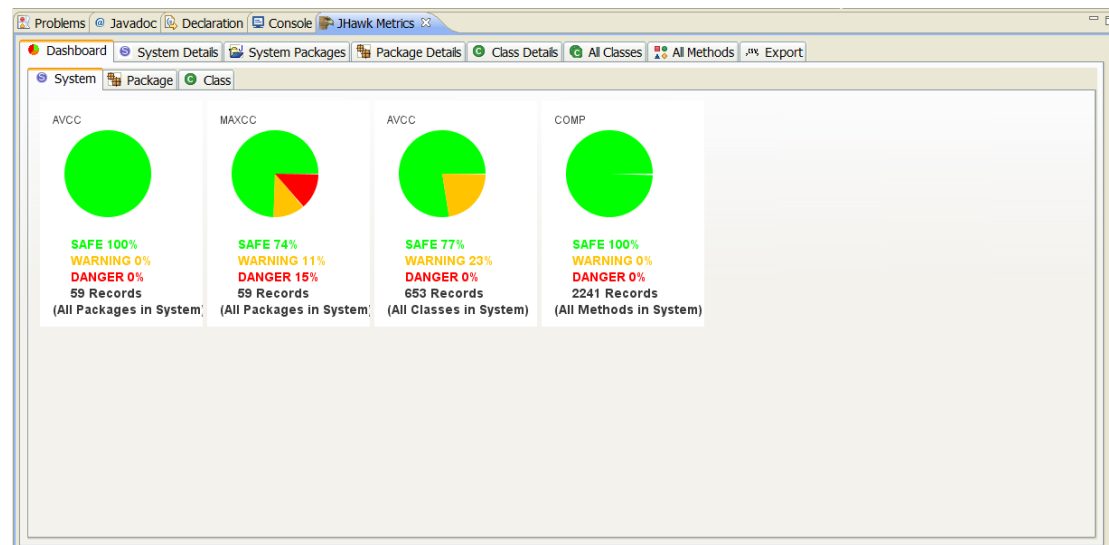
You can get guidance on setting eclipse memory here -

[http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.user/tasks/running\\_eclipse.htm](http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.user/tasks/running_eclipse.htm)

Eclipse is quite inefficient in its use of memory and if you want to process very large numbers of Java files with JHawk we would strongly recommend using the standalone version. The plugin has the benefit of being simple to use and close to the working code but it is much slower than the stand alone application.

### ***The JHawk Metrics view***

After the code has been analysed the JHawk Metrics View will be displayed in the bottom right hand tab group with the name 'JHawk Metrics' and the hawk icon. The tabs will be open on the dashboard tab.

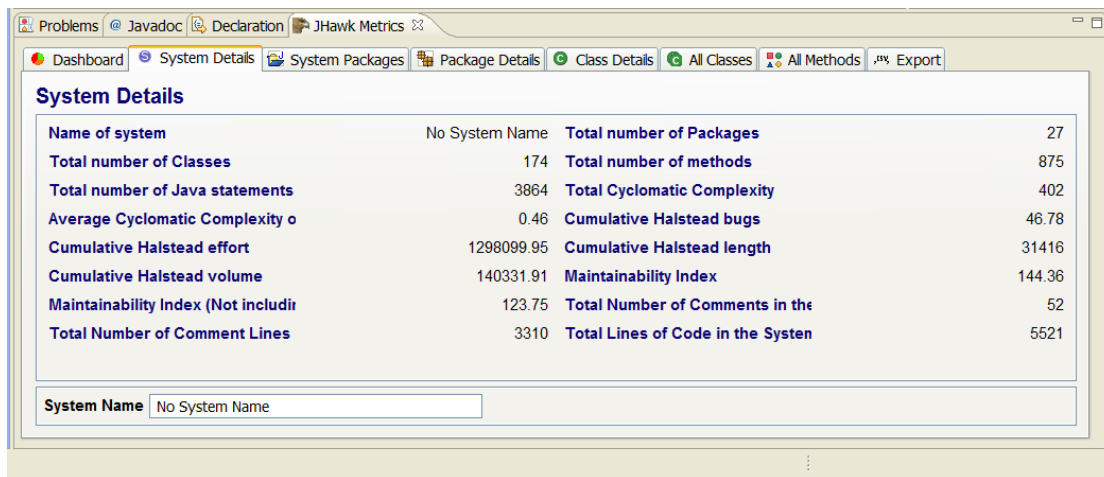


### ***The Dashboard tab***

The dashboard tab provides you with a visual summary of the state of your code at System, Package and Class level. It does this with a series of pie charts that reflect the proportion of elements (packages, classes, methods) whose metrics have crossed a particular warning or danger level. The default settings will be displayed when you first start the product. You can modify these to provide you with the diagnostic information that you need and you can reset the dashboard back to the default values at any time. You can find out how to do this by consulting the separate section on the on the Dashboard sub-tab elsewhere in the document.

## The System details tab

The System tab provides the ‘top level’ figures on your code. It shows you the main system level metrics relating to the code that you have chosen to analyse.



The screenshot shows the 'System Details' tab in the JHawk Metrics application. The interface includes a toolbar with tabs for Dashboard, System Details (selected), System Packages, Package Details, Class Details, All Classes, All Methods, and Export. The main content area displays a table of system-level metrics. At the bottom, there is a text input field for the 'System Name'.

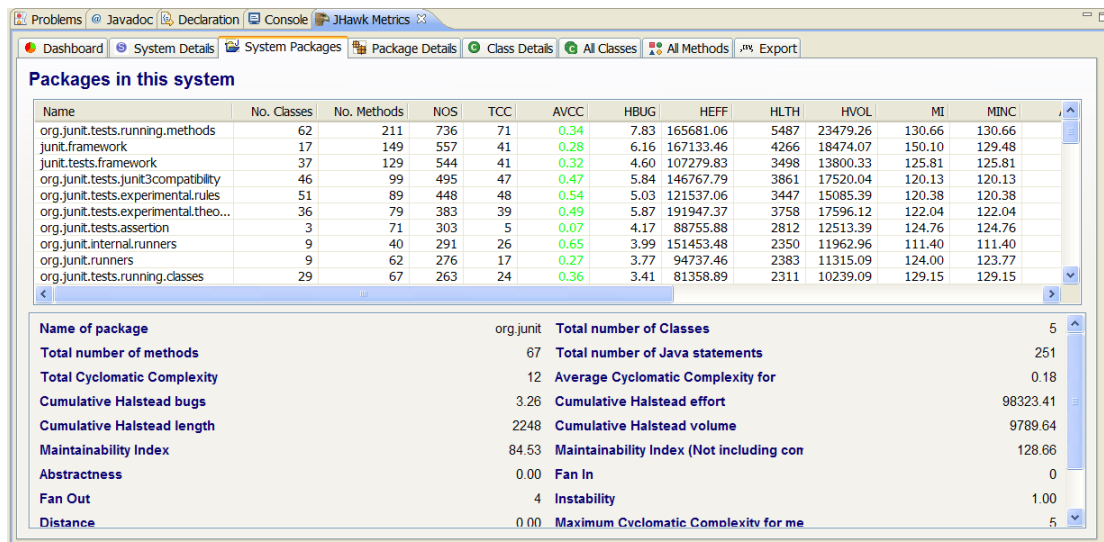
| System Details                                 |                |
|--|----------------|
| Name of system                                 | No System Name |
| Total number of Packages                       | 27             |
| Total number of Classes                        | 174            |
| Total number of methods                        | 875            |
| Total number of Java statements                | 3864           |
| Total Cyclomatic Complexity                    | 402            |
| Average Cyclomatic Complexity                  | 0.46           |
| Cumulative Halstead bugs                       | 46.78          |
| Cumulative Halstead effort                     | 1298099.95     |
| Cumulative Halstead length                     | 31416          |
| Cumulative Halstead volume                     | 140331.91      |
| Maintainability Index                          | 144.36         |
| Maintainability Index (Not including comments) | 123.75         |
| Total Number of Comments in the System         | 52             |
| Total Lines of Code in the System              | 5521           |

System Name:

The ‘System Name’ text edit allows you to enter a name that can be used to refer to the system that you are analysing.

## The System Packages tab

This is the first of a series of inter-related tabs that allow you to drill down through your data at System, Package, Class and Method level. These tabs all follow the same pattern – the top part of the tab is a list of all the artefacts at that level with the metrics collected summarised in the columns of the table. Clicking on a row in the table will cause the full detail relating to that artefact (in this case a Package) to be displayed in the area below.



The screenshot shows the 'System Packages' tab in the JHawk Metrics application. The interface includes a toolbar with tabs for Dashboard, System Details, System Packages (selected), Package Details, Class Details, All Classes, All Methods, and Export. The main content area displays a table of packages in the system. Below the table, there is a detailed view of the selected package 'org.junit'.

| Packages in this system             |             |             |     |     |      |      |           |      |          |        |        |
|-------------------------------------|-------------|-------------|-----|-----|------|------|-----------|------|----------|--------|--------|
| Name                                | No. Classes | No. Methods | NOS | TCC | AVCC | HBUG | HEFF      | HLTH | HVOL     | MI     | MINC   |
| org.junit.tests.running.methods     | 62          | 211         | 736 | 71  | 0.34 | 7.83 | 165681.06 | 5487 | 23479.26 | 130.66 | 130.66 |
| junit.framework                     | 17          | 149         | 557 | 41  | 0.28 | 6.16 | 167133.46 | 4266 | 18474.07 | 150.10 | 129.48 |
| junit.tests.framework               | 37          | 129         | 544 | 41  | 0.32 | 4.60 | 107279.83 | 3498 | 13800.33 | 125.81 | 125.81 |
| org.junit.tests.junit3compatibility | 46          | 99          | 495 | 47  | 0.47 | 5.84 | 146767.79 | 3861 | 17520.04 | 120.13 | 120.13 |
| org.junit.tests.experimental.rules  | 51          | 89          | 448 | 48  | 0.54 | 5.03 | 121537.06 | 3447 | 15085.39 | 120.38 | 120.38 |
| org.junit.tests.experimental.theory | 36          | 79          | 383 | 39  | 0.49 | 5.87 | 191947.37 | 3758 | 17596.12 | 122.04 | 122.04 |
| org.junit.tests.assertion           | 3           | 71          | 303 | 5   | 0.07 | 4.17 | 88755.88  | 2812 | 12513.39 | 124.76 | 124.76 |
| org.junit.internal.runners          | 9           | 40          | 291 | 26  | 0.65 | 3.99 | 151453.48 | 2350 | 11962.96 | 111.40 | 111.40 |
| org.junit.runners                   | 9           | 62          | 276 | 17  | 0.27 | 3.77 | 94737.46  | 2383 | 11315.09 | 124.00 | 123.77 |
| org.junit.tests.running.classes     | 29          | 67          | 263 | 24  | 0.36 | 3.41 | 81358.89  | 2311 | 10239.09 | 129.15 | 129.15 |

Selected Package: org.junit

| Package Details                                |           |
|--|-----------|
| Name of package                                | org.junit |
| Total number of Classes                        | 5         |
| Total number of methods                        | 67        |
| Total number of Java statements                | 251       |
| Total Cyclomatic Complexity                    | 12        |
| Average Cyclomatic Complexity for              | 0.18      |
| Cumulative Halstead bugs                       | 3.26      |
| Cumulative Halstead effort                     | 98323.41  |
| Cumulative Halstead length                     | 2248      |
| Cumulative Halstead volume                     | 9789.64   |
| Maintainability Index                          | 84.53     |
| Maintainability Index (Not including comments) | 128.66    |
| Abstractness                                   | 0.00      |
| Fan In   | 0         |
| Fan Out  | 4         |
| Instability                                    | 1.00      |
| Distance                                       | 0.00      |
| Maximum Cyclomatic Complexity for methods      | 5         |

Let us say that you noted that the Average Cyclomatic Complexity was quite high and you wanted to find out which code was responsible for this. Obviously packages with a higher level of average cyclomatic complexity will contribute more to this final figure so you need to look at the Packages list. By double clicking on the AVCC (Average Cyclomatic Complexity) column header you will see that the list of packages re-sorts itself in ascending order based on AVCC. Double clicking again will bring the list into descending order based on AVCC with the highest level at the top.

You can now see which package has the highest Average Cyclomatic Complexity. We now need to see which classes contribute most in this package. We can do selecting the package with the highest value. This will cause the 'Package Details' tab to be populated with data relating to this package. To proceed with our analysis we can click on the 'Package Details' tab.

### The Package Details tab.

At the top of this tab we find a list of classes contained in the package. Clicking on a Class will display full details of the metrics for the class. Double clicking on a class in the table will cause that classes source file to be opened in an editor on the workbench.

To continue our example we can double click on the Average Cyclomatic Complexity (AVCC) column to find the Class with the highest Average Cyclomatic complexity value.

**Classes in package org.junit**

| Name                             | Superclass               | No. ... | LOCOM | TCC | MAXCC | AVCC | NOS | HBUG | HEFF     | HLTH | HVOL    |
|----------------------------------|--------------------------|---------|-------|-----|-------|------|-----|------|----------|------|---------|
| Assert                           | java.lang.Object         | 50      | 0.00  | 66  | 5     | 1.32 | 123 | 1.68 | 42903.55 | 1221 | 5049.48 |
| Assume                           | java.lang.Object         | 4       | 0.00  | 5   | 2     | 1.25 | 9   | 0.11 | 2040.36  | 85   | 328.19  |
| ComparisonFailure                | java.lang.AssertionError | 4       | 0.75  | 4   | 1     | 1.00 | 16  | 0.13 | 1745.43  | 100  | 394.72  |
| ComparisonFailure\$Comparison... | java.lang.Object         | 8       | 0.70  | 17  | 3     | 2.12 | 49  | 0.62 | 18977.56 | 410  | 1852.62 |
| None                             | java.lang.Throwable      | 1       | 1.00  | 1   | 1     | 1.00 | 3   | 0.02 | 234.03   | 19   | 63.51   |

**Name of class** Assert **No. of External Methods called** org.hamcrest.StringDescription.String

**No. of Methods called that are in the cl:**  **No. of local methods called** assertNotSame (1)

**No. of instance variables declared**  **No. of interfaces implemented**

**No. of packages imported** org.hamcrest.Description **Name of superclass** java.lang.Object

**Total number of methods** 50 **Lack of Cohesion of methods** 0.00

**Total Cyclomatic Complexity** 66 **Maximum Cyclomatic Complexity** 5

**Average Cyclomatic Complexity** 1.32 **Total number of Java statements (alterr** 123

We can then select the class with the highest Average Cyclomatic complexity value. This class will be used to populate the next tab – the 'Class details' tab.

### The Class Details tab

In a similar pattern to the tabs before we find a table of methods and a summary of the metrics associated with each method displayed in the columns.

**Methods in class Assert**

| Name                                   | COMP | NOA | NOCL | NOC | VDEC | VREF | NOS | NEXP | MDN | HLTH | HVOC | HVOL   | HDIF  | HEFF    |
|--|------|-----|------|-----|------|------|-----|------|-----|------|------|--------|-------|---------|
| assertEquals(java.lang.String, ja...   | 5    | 3   | 14   | 1   | 1    | 19   | 6   | 14   | 1   | 76   | 30   | 372.92 | 24.19 | 9021.88 |
| assertEquals(java.lang.String, do...   | 3    | 4   | 19   | 1   | 0    | 8    | 3   | 11   | 0   | 51   | 27   | 242.50 | 12.07 | 2927.31 |
| format(java.lang.String, java.lan...   | 3    | 3   | 0    | 0   | 3    | 16   | 6   | 16   | 0   | 90   | 33   | 454.00 | 15.28 | 6934.78 |
| assertTrue(java.lang.String, bool...   | 2    | 2   | 10   | 1   | 0    | 2    | 2   | 2    | 0   | 18   | 14   | 68.53  | 4.50  | 308.40  |
| fail(java.lang.String)(org.junit.As... | 2    | 1   | 8    | 1   | 0    | 2    | 2   | 3    | 0   | 20   | 17   | 81.75  | 6.43  | 525.53  |
| assertSame(java.lang.String, jav...    | 2    | 3   | 12   | 1   | 0    | 5    | 3   | 4    | 0   | 25   | 16   | 100.00 | 7.00  | 700.00  |
| assertNotSame(java.lang.String, ...    | 2    | 3   | 13   | 1   | 0    | 3    | 2   | 2    | 0   | 21   | 15   | 82.04  | 5.25  | 430.73  |
| failSame(java.lang.String)(org.ju...   | 2    | 1   | 0    | 0   | 1    | 4    | 4   | 4    | 0   | 30   | 19   | 127.44 | 6.75  | 860.21  |
| failNotSame(java.lang.String, jav...   | 2    | 3   | 0    | 0   | 1    | 6    | 4   | 4    | 0   | 42   | 24   | 192.57 | 6.90  | 1328.72 |
| formatClassAndValue(java.lang....      | 2    | 2   | 0    | 0   | 1    | 4    | 3   | 3    | 0   | 36   | 24   | 165.06 | 9.00  | 1485.53 |

**Name of method** assertArrayEquals **Number of arguments** expecteds (int[])

**Number of Variable declarations**  **Number of Variable references** actuals (1)

**Number of classes referenced**  **Number of methods external to this cla**

**Number of methods local to this class c** assertArrayEquals (1) **Number of Exceptions referenced**

**Number of exceptions thrown**  **Cyclomatic Complexity** 1

**Number of comment lines** 9 **Number of comments** 1

Selecting a method will populate the lower display pane. Double clicking on a method in the table will cause that source file of the class that contains the method to be opened in an editor on the workbench.

As in the previous step, we now perform a similar exercise with the methods as we did with the Classes.

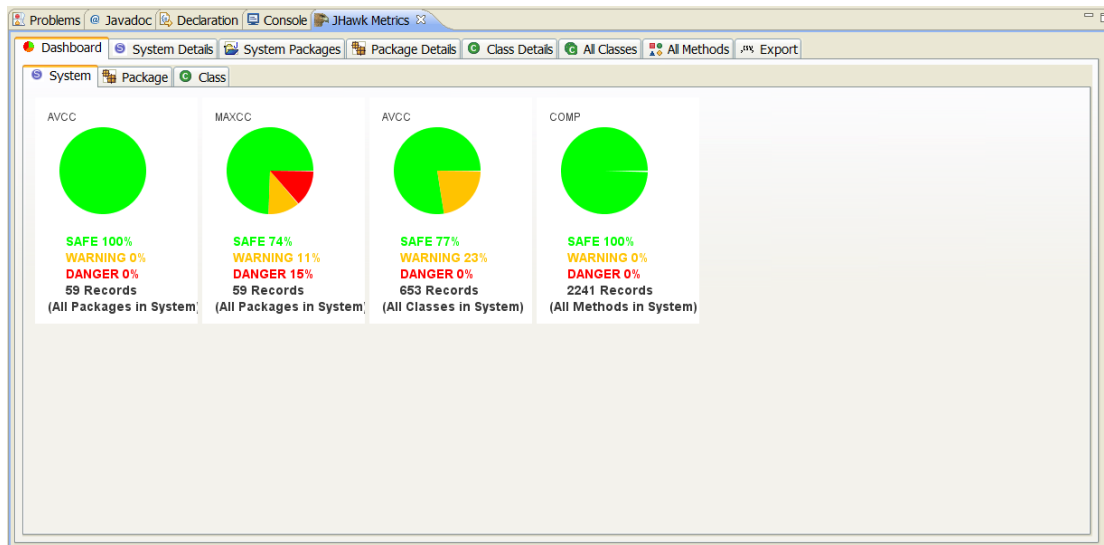
In this case we are interested in the actual Cyclomatic Complexity of each method. Double clicking on the COMP header will list the methods in order of COMP value. As before, double clicking a second time will sort the methods in descending order. It is now easy to see the methods with the highest COMP values.

As you can probably guess you can do this for any metric and using the above approach find where your problem code lies.

If you are looking for issues that are at Method or Class level you can view and sort all the Classes and Methods at once by metric using the 'All Classes in System' and 'All Methods In System' tabs respectively. Double clicking on any of the entries in these tables will cause the associated source file to be opened in an editor on the Workbench.

This has been a brief introduction to the main screens of the application. To find out more about how to configure JHawk to better suit your purposes you should look at the 'The Preferences Tab' later in this guide. If you want to find out how to export data for use externally then you should read the 'The Export Tab' section.

# The Dashboard Tab

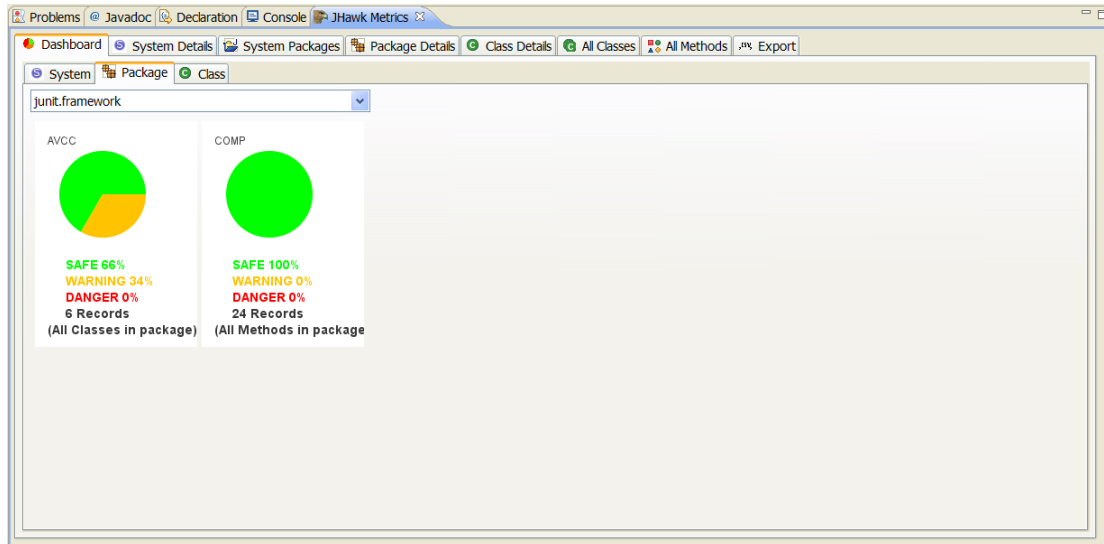


For users familiar with previous versions of JHawk you will note that this is a new tab. The dashboard sub-tab allows users to make a quick visual check on the current quality of their code. Although it comes with some predefined metrics the dashboard tab is completely configurable using Preferences (see 'Preferences – Configuring the Dashboard' for more details).

Each metric is displayed as a pie chart with the proportion of values that are OK shown as a green slice, the proportion that exceed the warning level shown as a yellow slice and the proportion exceeding the danger level shown as a red slice. These are the default colours used – if you have changed the colours using the General Preferences tab then these will be different. The name of the metric calculated, the number of entries used in the calculation and the percentage in each category are also displayed.

There are three sub-tabs within the Dashboard Tab –

- The System tab shows all the metrics that have been assigned to the dashboard at package, class and method level. Package level metrics will be displayed for all the packages in the system, class level metrics for all the classes in the system and method level metrics for all the methods in the system.
- The Package tab shows all the metrics assigned to the dashboard at class and method level. A dropdown list at the top of the pane allows you to select the package for which you wish the data to be shown.
- The Class tab shows all the metrics assigned to the dashboard at method level. A dropdown list at the top of the pane allows you to select the class for which you wish the data to be shown.



In both the Package and Class sub tabs the drop down lists of sub-elements are ordered according to whether they have any metrics in the danger category, then in the warning category then in the safe category. Note that only metrics selected for the dashboard will be considered when setting the order of the selected item.

# System Details Tab

|  |                |   |        |
|--|----------------|---|--------|
| <b>System Details</b>  |                |   |        |
| <b>Name of system</b>  | No System Name | <b>Total number of Packages</b>               | 27     |
| <b>Total number of Classes</b>                                 | 174            | <b>Total number of methods</b>                | 875    |
| <b>Total number of Java statements</b>                         | 3864           | <b>Total Cyclomatic Complexity</b>            | 402    |
| <b>Average Cyclomatic Complexity</b>                           | 0.46           | <b>Cumulative Halstead bugs</b>               | 46.78  |
| <b>Cumulative Halstead effort</b>                              | 1298099.95     | <b>Cumulative Halstead length</b>             | 31416  |
| <b>Cumulative Halstead volume</b>                              | 140331.91      | <b>Maintainability Index</b>                  | 144.36 |
| <b>Maintainability Index (Not including comments)</b>          | 123.75         | <b>Total Number of Comments in the System</b> | 52     |
| <b>Total Number of Comment Lines</b>                           | 3310           | <b>Total Lines of Code in the System</b>      | 5521   |
| <b>System Name</b> <input type="text" value="No System Name"/> |                |   |        |

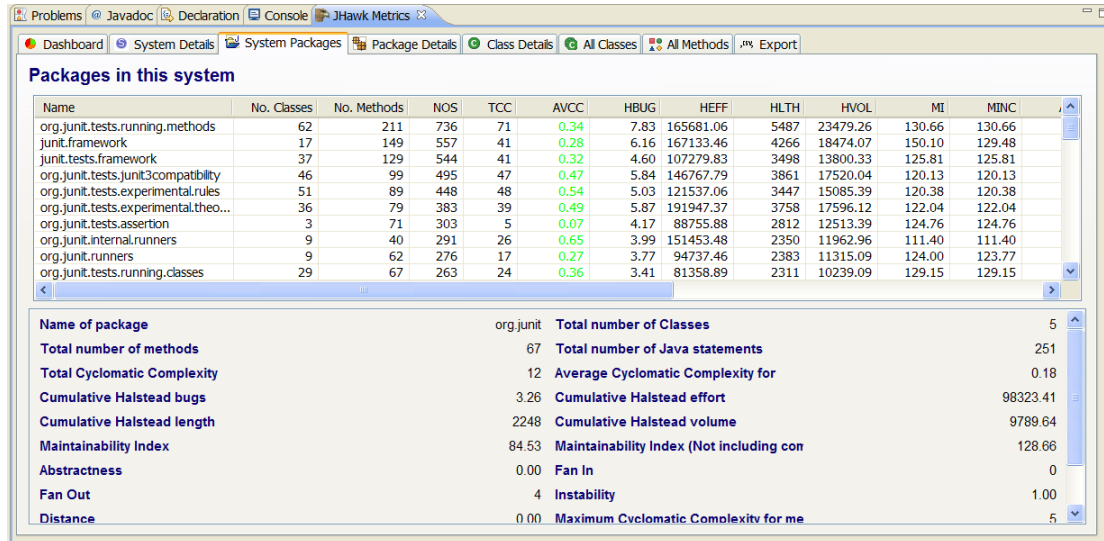
The System Details tab summarises metrics data at the System level.

The 'System Name' text edit allows you to enter a name that can be used to refer to the system that you are analysing. The default value is 'No System Name'.



# The System Packages Tab

The System packages tab is divided horizontally into two panels – the top panel shows a list of the packages in the system with the metrics collected in each column. Only the active metrics are shown. You can configure the metrics shown in the table by using the Preferences tab (see the section – ‘Preferences Tab – Configuring Metrics’). You can see all the metrics collected for an individual package by selecting it – all the metrics for that package will be displayed in the panel below.



**Packages in this system**

| Name                                | No. Classes | No. Methods | NOS | TCC | AVCC | HBUG | HEFF      | HLTH | HVOL     | MI     | MINC   |
|-------------------------------------|-------------|-------------|-----|-----|------|------|-----------|------|----------|--------|--------|
| org.junit.tests.running.methods     | 62          | 211         | 736 | 71  | 0.34 | 7.83 | 165681.06 | 5487 | 23479.26 | 130.66 | 130.66 |
| junit.framework                     | 17          | 149         | 557 | 41  | 0.28 | 6.16 | 167133.46 | 4266 | 18474.07 | 150.10 | 129.48 |
| junit.tests.framework               | 37          | 129         | 544 | 41  | 0.32 | 4.60 | 107279.83 | 3498 | 13800.33 | 125.81 | 125.81 |
| org.junit.tests.junit3compatibility | 46          | 99          | 495 | 47  | 0.47 | 5.84 | 146767.79 | 3861 | 17520.04 | 120.13 | 120.13 |
| org.junit.tests.experimental.rules  | 51          | 89          | 448 | 48  | 0.54 | 5.03 | 121537.06 | 3447 | 15085.39 | 120.38 | 120.38 |
| org.junit.tests.experimental.theory | 36          | 79          | 383 | 39  | 0.49 | 5.87 | 191947.37 | 3758 | 17596.12 | 122.04 | 122.04 |
| org.junit.tests.assertion           | 3           | 71          | 303 | 5   | 0.07 | 4.17 | 88755.88  | 2812 | 12513.39 | 124.76 | 124.76 |
| org.junit.internal.runners          | 9           | 40          | 291 | 26  | 0.65 | 3.99 | 151453.48 | 2350 | 11962.96 | 111.40 | 111.40 |
| org.junit.runners                   | 9           | 62          | 276 | 17  | 0.27 | 3.77 | 94737.46  | 2383 | 11315.09 | 124.00 | 123.77 |
| org.junit.tests.running.classes     | 29          | 67          | 263 | 24  | 0.36 | 3.41 | 81358.89  | 2311 | 10239.09 | 129.15 | 129.15 |

|                                    |           |   |          |
|------------------------------------|-----------|---|----------|
| <b>Name of package</b>             | org.junit | <b>Total number of Classes</b>                  | 5        |
| <b>Total number of methods</b>     | 67        | <b>Total number of Java statements</b>          | 251      |
| <b>Total Cyclomatic Complexity</b> | 12        | <b>Average Cyclomatic Complexity for</b>        | 0.18     |
| <b>Cumulative Halstead bugs</b>    | 3.26      | <b>Cumulative Halstead effort</b>               | 98323.41 |
| <b>Cumulative Halstead length</b>  | 2248      | <b>Cumulative Halstead volume</b>               | 9789.64  |
| <b>Maintainability Index</b>       | 84.53     | <b>Maintainability Index (Not including con</b> | 128.66   |
| <b>Abstractness</b>                | 0.00      | <b>Fan In</b>                                   | 0        |
| <b>Fan Out</b>                     | 4         | <b>Instability</b>                              | 1.00     |
| <b>Distance</b>                    | 0.00      | <b>Maximum Cyclomatic Complexity for me</b>     | 5        |

Selecting a Package in this tab will set the package details to be displayed in the ‘Package Details’ tab.

# The Package Details tab

The screenshot shows the JHawk Metrics application interface. The 'Package Details' tab is active, displaying a table of classes in the package 'org.junit'. Below the table, a detailed view for the 'Assert' class is shown, including various metrics and configuration options.

| Name                             | Superclass               | No. ... | LCOM | TCC | MAXCC | AVCC | NOS | HBUG | HEFF     | HLTH | HVOL    |
|----------------------------------|--------------------------|---------|------|-----|-------|------|-----|------|----------|------|---------|
| Assert                           | java.lang.Object         | 50      | 0.00 | 66  | 5     | 1.32 | 123 | 1.68 | 42903.55 | 1221 | 5049.48 |
| Assume                           | java.lang.Object         | 4       | 0.00 | 5   | 2     | 1.25 | 9   | 0.11 | 2040.36  | 85   | 328.19  |
| ComparisonFailure                | java.lang.AssertionError | 4       | 0.75 | 4   | 1     | 1.00 | 16  | 0.13 | 1745.43  | 100  | 394.72  |
| ComparisonFailure\$Comparison... | java.lang.Object         | 8       | 0.70 | 17  | 3     | 2.12 | 49  | 0.62 | 18977.56 | 410  | 1852.62 |
| None                             | java.lang.Throwable      | 1       | 1.00 | 1   | 1     | 1.00 | 3   | 0.02 | 234.03   | 19   | 63.51   |

**Class Details for Assert:**

- Name of class: Assert
- No. of External Methods called: org.hamcrest.StringDescription.String
- No. of Methods called that are in the class: [Dropdown]
- No. of local methods called: assertNotSame (1)
- No. of instance variables declared: [Dropdown]
- No. of interfaces implemented: [Dropdown]
- No. of packages imported: org.hamcrest.Description
- Name of superclass: java.lang.Object
- Total number of methods: 50
- Lack of Cohesion of methods: 0.00
- Total Cyclomatic Complexity: 66
- Maximum Cyclomatic Complexity: 5
- Average Cyclomatic Complexity: 1.32
- Total number of Java statements (altered): 123

The Package selected in the System Packages tab will be the Package used in the Package Details tab. The Package Details tab is divided horizontally into two panels – the top panel displays a table showing a list of the classes in the package with the metrics collected in each column. Only the active metrics are shown. You can configure the metrics shown in the table by using the Preferences tab (see the section – ‘Preferences Tab – Configuring Metrics’). You can see all the metrics collected for an individual class by clicking on the listing for that class in the table – the metrics will then be displayed in the panel below.

Double clicking on any class in the table will cause the associated source file to be opened in an editor on the Workbench.

# The Class Details tab

**Methods in class Assert**

| Name                                   | COMP | NOA | NOCL | NOC | VDEC | VREF | NOS | NEXP | MDN | HLTH | HVOC | HVOL   | HDIF  | HEFF    |
|--|------|-----|------|-----|------|------|-----|------|-----|------|------|--------|-------|---------|
| assertEquals(java.lang.String, ja...   | 5    | 3   | 14   | 1   | 1    | 19   | 6   | 14   | 1   | 76   | 30   | 372.92 | 24.19 | 9021.88 |
| assertEquals(java.lang.String, do...   | 3    | 4   | 19   | 1   | 0    | 8    | 3   | 11   | 0   | 51   | 27   | 242.50 | 12.07 | 2927.31 |
| format(java.lang.String, java.lan...   | 3    | 3   | 0    | 0   | 3    | 16   | 6   | 16   | 0   | 90   | 33   | 454.00 | 15.28 | 6934.78 |
| assertTrue(java.lang.String, bool...   | 2    | 2   | 10   | 1   | 0    | 2    | 2   | 2    | 0   | 18   | 14   | 68.53  | 4.50  | 308.40  |
| fail(java.lang.String)(org.junit.As... | 2    | 1   | 8    | 1   | 0    | 2    | 2   | 3    | 0   | 20   | 17   | 81.75  | 6.43  | 525.53  |
| assertSame(java.lang.String, jav...    | 2    | 3   | 12   | 1   | 0    | 5    | 3   | 4    | 0   | 25   | 16   | 100.00 | 7.00  | 700.00  |
| assertNotSame(java.lang.String, ...    | 2    | 3   | 13   | 1   | 0    | 3    | 2   | 2    | 0   | 21   | 15   | 82.04  | 5.25  | 430.73  |
| failSame(java.lang.String)(org.Ju...   | 2    | 1   | 0    | 0   | 1    | 4    | 4   | 4    | 0   | 30   | 19   | 127.44 | 6.75  | 860.21  |
| failNotSame(java.lang.String, jav...   | 2    | 3   | 0    | 0   | 1    | 6    | 4   | 4    | 0   | 42   | 24   | 192.57 | 6.90  | 1328.72 |
| formatClassAndValue(java.lang....      | 2    | 2   | 0    | 0   | 1    | 4    | 3   | 3    | 0   | 36   | 24   | 165.06 | 9.00  | 1485.53 |

**Name of method** assertArrayEquals **Number of arguments** expecteds (int[])

**Number of Variable declarations**  **Number of Variable references** actuals (1)

**Number of classes referenced**  **Number of methods external to this cla**

**Number of methods local to this class** assertArrayEquals (1) **Number of Exceptions referenced**

**Number of exceptions thrown**  **Cyclomatic Complexity** 1

**Number of comment lines** 9 **Number of comments** 1

The Class selected in the Package details tab will be the Class used in the Class Details tab. The Class Details tab displays a table showing a list of the methods in the class with the metrics collected in each column. Only the active metrics are shown. You can configure the metrics shown in the table by using the Preferences tab (see the section – ‘Preferences Tab – Configuring Metrics’). You can see all the metrics collected for an individual method by clicking on the listing for that method in the table – the metrics will be displayed in the panel below.

Double clicking on any method in the table will cause the associated source file to be opened in an editor on the Workbench.

# The All Classes Tab

| Name                | Superclass               | No. ... | LCOM | TCC | MAXCC | AVCC | NOS | HBUG | HEFF     | HLTH | HVOL    |
|---------------------|--------------------------|---------|------|-----|-------|------|-----|------|----------|------|---------|
| Is                  | org.hamcrest.BaseMatcher | 6       | 0.80 | 7   | 2     | 1.17 | 14  | 0.18 | 3829.16  | 144  | 554.67  |
| RunWith             | java.lang.Object         | 7       | 3.00 | 8   | 2     | 1.14 | 23  | 0.54 | 14408.89 | 362  | 1617.56 |
| WithDataPointMethod | java.lang.Object         | 8       | 0.00 | 9   | 2     | 1.12 | 22  | 0.33 | 5087.20  | 242  | 985.73  |
| TextFeedbackTest    | junit.framework.TestCase | 8       | 0.29 | 9   | 2     | 1.12 | 51  | 0.57 | 10566.49 | 372  | 1697.36 |
| MaxStarterTest      | java.lang.Object         | 20      | 0.74 | 22  | 2     | 1.10 | 121 | 1.84 | 40795.35 | 1141 | 5530.84 |
| TimeoutTest         | java.lang.Object         | 10      | 0.00 | 11  | 2     | 1.10 | 67  | 0.97 | 20576.25 | 604  | 2902.65 |
| RunNotifier         | java.lang.Object         | 11      | 0.85 | 12  | 2     | 1.09 | 27  | 0.22 | 2908.57  | 179  | 672.73  |
| TestCaseTest        | junit.framework.TestCase | 17      | 0.00 | 18  | 2     | 1.06 | 73  | 0.61 | 8859.02  | 454  | 1829.20 |
| None                | java.lang.Throwable      | 1       | 1.00 | 1   | 1     | 1.00 | 3   | 0.02 | 234.03   | 19   | 63.51   |
| ComparisonFailure   | java.lang.AssertionError | 4       | 0.75 | 4   | 1     | 1.00 | 16  | 0.13 | 1745.43  | 100  | 394.72  |
| WasRun              | junit.framework.TestCase | 1       | 0.00 | 1   | 1     | 1.00 | 4   | 0.03 | 284.37   | 25   | 86.49   |
| AllTests            | java.lang.Object         | 2       | 0.00 | 2   | 1     | 1.00 | 9   | 0.12 | 3361.35  | 86   | 355.96  |

| Name of class                             |                          | No. of External Methods called          |                     |
|---|--------------------------|---|---------------------|
| WithDataPointMethod                       |                          | .potentialsForNextUnassigned (1)        |                     |
| No. of Methods called that are in the cla |                          | No. of local methods called             | potentialValues (2) |
| No. of instance variables declared        |                          | No. of interfaces implemented           |                     |
| No. of packages imported                  | java.lang.reflect.Method | Name of superclass                      | java.lang.Object    |
| Total number of methods                   | 8                        | Lack of Cohesion of methods             | 0.00                |
| Total Cyclomatic Complexity               | 9                        | Maximum Cyclomatic Complexity           | 2                   |
| Average Cyclomatic Complexity             | 1.12                     | Total number of Java statements (altern | 22                  |
| Cumulative Halstead lines                 | 0.33                     | Cumulative Halstead effort              | 5087.20             |

By default the All Classes in System Tab is not enabled – this is because it expensive to create in terms of memory and processing power and for Systems containing a very large number of classes it could slow down the analysis process. To enable the All Classes in System tab you need to use the Preferences Tab and go the to the ‘General’ sub-tab. See the section ‘Preferences Tab– General Preferences’.

The All Classes tab consists of a table showing a list of all the classes in the System with the metrics collected in each column. Only the active metrics are shown. You can configure the metrics shown in the table by using the Preferences tab (see the section – ‘Preferences Tab – Configuring Metrics’). You can see all the metrics collected for an individual class by clicking on the listing for that class in the table – details of the metrics will be displayed in the panel below.

Double clicking on any class in the table will cause the associated source file to be opened in an editor on the Workbench.

# All Methods Tab

The screenshot shows the JHawk Metrics application with the 'All Methods' tab selected. The top table lists various methods with their metrics. Below it, the 'appendValue' method is selected, and its details are shown in a form.

| Name                                   | COMP | NOA | NOCL | NOC | VDEC | VREF | NOS | NEXP | MDN | HLTH | HVOC | HVOL     | HDIF  | HEFF     | H |
|--|------|-----|------|-----|------|------|-----|------|-----|------|------|----------|-------|----------|---|
| getTest(java.lang.String)(junit.ru...  | 11   | 1   | 6    | 3   | 4    | 23   | 31  | 35   | 1   | 215  | 56   | 1248.... | 22.85 | 28531.77 | ( |
| start(java.lang.String[])(junit.tex... | 9    | 1   | 4    | 1   | 6    | 34   | 20  | 50   | 2   | 208  | 63   | 1243.... | 22.33 | 27756.82 | ( |
| createTest(java.lang.Class, java.l...  | 9    | 2   | 4    | 1   | 2    | 23   | 16  | 27   | 2   | 174  | 54   | 1001.... | 22.91 | 22940.03 | ( |
| appendValue(java.lang.Object)(...      | 8    | 1   | 0    | 0   | 0    | 21   | 28  | 34   | 1   | 154  | 39   | 813.95   | 12.96 | 10552.31 | ( |
| TestSuite(java.lang.Class)(junit.fr... | 8    | 1   | 7    | 2   | 2    | 11   | 17  | 20   | 1   | 144  | 56   | 836.26   | 19.86 | 16611.15 | ( |
| TestClass(java.lang.Class)(org.ju...   | 7    | 1   | 6    | 1   | 0    | 5    | 11  | 14   | 1   | 79   | 39   | 417.55   | 11.48 | 4792.71  | ( |
| arrayEquals(java.lang.String, jav...   | 7    | 3   | 16   | 1   | 4    | 24   | 18  | 29   | 3   | 129  | 48   | 720.46   | 27.50 | 19812.65 | ( |
| makeDescription(junit.framework...     | 7    | 1   | 1    | 1   | 8    | 27   | 24  | 29   | 2   | 173  | 47   | 960.94   | 21.82 | 20964.59 | ( |
| equals(java.lang.Object)(junit.sa...   | 7    | 1   | 0    | 0   | 1    | 13   | 8   | 14   | 1   | 80   | 30   | 392.55   | 14.44 | 5667.46  | ( |

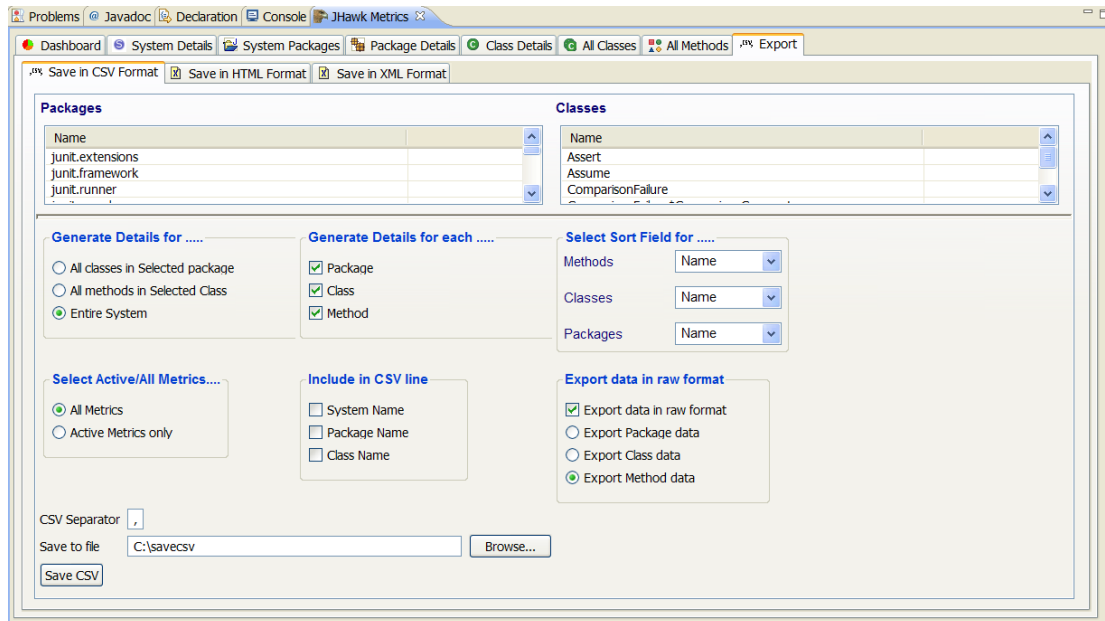
| Name of method                        |                      | appendValue                             |  | Number of arguments                     |  | value (java.lang.Object) |  |
|---------------------------------------|----------------------|---|--|---|--|--------------------------|--|
| Number of Variable declarations       |                      | Number of Variable references           |  | String (2)                              |  |                          |  |
| Number of classes referenced          | java.lang.String (1) | Number of methods external to this clas |  | InternalArrayIterator ArrayIterator (1) |  |                          |  |
| Number of methods local to this class | valueOf (4)          | Number of Exceptions referenced         |  |   |  |                          |  |
| Number of exceptions thrown           |                      | Cyclomatic Complexity                   |  | 8                                       |  |                          |  |
| Number of comment lines               |                      | Number of comments                      |  | 0                                       |  |                          |  |
| Number of Java statements             | 28                   | Number of Java Expressions              |  | 34                                      |  |                          |  |
| Maximum depth of nesting              | 1                    | Halstead length                         |  | 154                                     |  |                          |  |
| Halstead Vocabulary                   | 39                   | Halstead Volume                         |  | 813.95                                  |  |                          |  |

By default the All Methods in System Tab is not enabled – this is because it expensive to create in terms of memory and processing power and for Systems containing a very large number of methods it could slow down the analysis process. To enable the All Methods in System tab you need to use the Preferences Tab and go the to the ‘General’ sub-tab. See the section ‘Preferences Tab– General Preferences’.

The All Methods tab consists of a table showing a list of all the methods in the System with the metrics collected in each column. Only the active metrics are shown. You can configure the metrics shown in the table by using the Preferences tab (see the section – ‘Preferences Tab – Configuring Metrics’). You can see all the metrics collected for an individual method by double clicking on the listing for that method in the table.

Double clicking on any method in the table will cause the associated source file to be opened in an editor on the Workbench.

# The Export Tab



The Export tab opens on the 'Save in CSV format' sub-tab, which is used to export data in CSV format. This is the first of three sub-tabs – the other two are the 'Save in HTML format' tab (which allows you to export data in HTML format), and the 'Save in XML format' tab (which allows you to export data in XML format, including the interchange format that is used in the JHawk Data Viewer).

## The CSV format tab

This Tab helps you to create a single CSV file containing the results of the analysis of the Java files that you have selected.

At the top of the screen you will see two lists which allow you to select packages and or classes whose data you wish to export. Below this to the left you will see a Box entitled 'Generate Details For...' and three radio buttons entitled 'All Classes in selected Package', 'All methods in Selected class' and 'Entire System'. These selections are related to the two lists above – obviously if you select 'Entire System' the selections in the two lists will be ignored and the details for every method in every class in every package in the system will be exported.

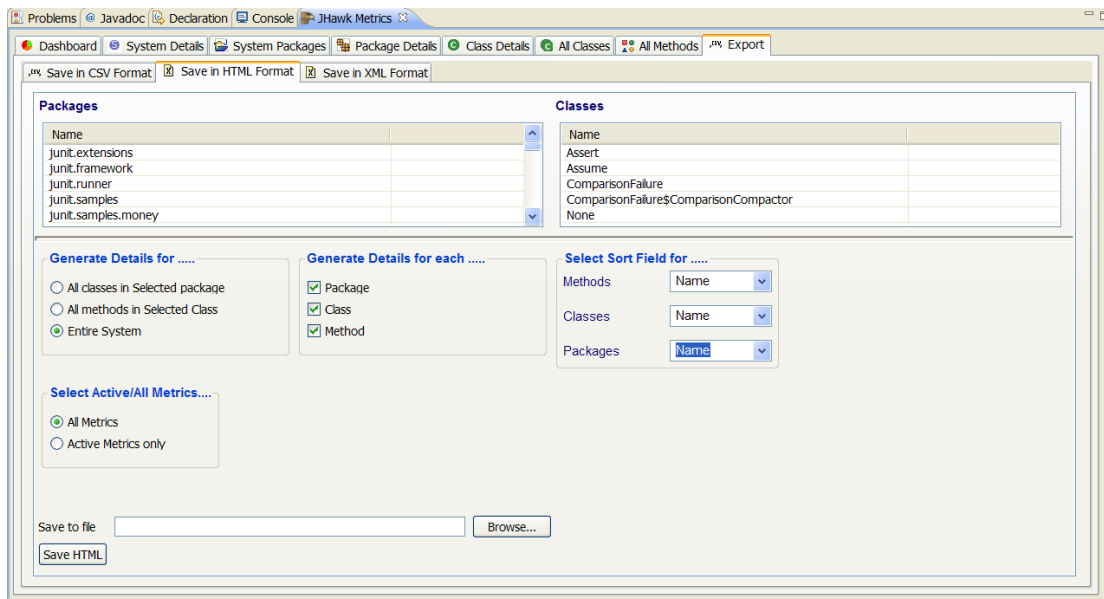
Next we come to a box entitled 'Generate Details for Each...' and selections for Package, Class and Method – as each of these is selected the 'Select Sort Field...' button for each level will be enabled. This allows you to select the field used to sort the data when it is exported to the CSV file. Clicking on the button opens a dialog that allows you to select the field and the order ('Ascending' or 'Descending') that the data is to be selected on.

On the next line on the left hand side we can select whether all the metrics are to be exported or only those metrics that are marked as active. To the right of this we can select which name fields will appear on the export line. At the furthest right is an option to allow us to export the data in 'raw' format. This simply means that the data is exported on the basis of one line per artefact at the level selected (Package, Class or Method) . No summary lines are produced with the raw format. Clicking on the raw check box will cause some of the other options to be disabled.

On the next line we can select the file name to which the data is to be exported. You can either enter a file name or use the browse button to select one.

On the bottom line we can select the separator to be used for the exported data (the default is ',').

## The HTML Tab



This Tab helps you to create a number of HTML files containing the results of the analysis of the Java files that you have selected. These files are constructed in a logical way from the main HTML file allowing you to ‘drill down’ through the results to find the data that you need.

At the top of the screen you will see two lists which allow to select packages and or classes whose data you wish to export. Below this to the left you will see a Box entitled ‘Generate Details For...’ and three radio buttons entitled ‘All Classes in selected Package’, ‘All methods in Selected class’ and ‘Entire System’. These selections are related to the two lists above – obviously if you select ‘Entire System’ the selections in the two lists will be ignored and the details for every method in every class in every package in the system will be exported.

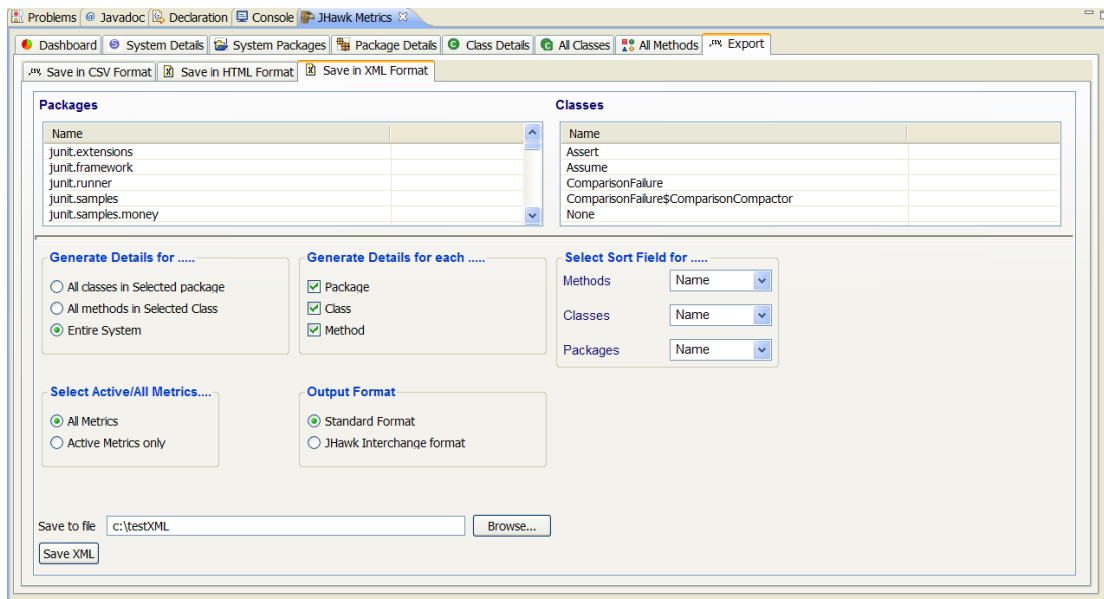
Next we come to a box entitled ‘Generate Details for Each...’ and selections for Package, Class and Method – as each of these is selected the ‘Select Sort Field...’ button for each level will be enabled. This allows you to select the field used to sort the data on when it is exported to the HTML file. Clicking on the button opens a dialog that allows you to select the field and the order (‘Ascending’ or ‘Descending’ that the data is to be selected on.

On the next line we can select whether all the metrics are to be exported or only those metrics that are marked as active.

The final line allows us to select the main HTML file which will act as the index file for all of the other HTML files created. All of the HTML files will be located in the same directory , or sub-directories, of the directory where the root HTML file is located.

## The Create XML Tab

This Tab helps you to create a single XML file containing the results of the analysis of the Java files that you have selected.



At the top of the screen you will see two lists which help you select packages and or classes whose data you wish to export. Below this to the left you will see a Box entitled ‘Generate Details For...’ and three radio buttons entitled ‘All Classes in selected Package’, ‘All methods in Selected class’ and ‘Entire System’. These selections are related to the two lists above – obviously if you select ‘Entire System’ the selections in the two lists will be ignored and the details for every method in every class in every package in the system will be exported.

Below this you will see a box entitled ‘Generate Details for Each...’ and selections for Package, Class and Method – as each of these is selected the ‘Select Sort Field...’ button for each level will be enabled. This allows you to select the field used to sort the data on when it is exported to the HTML file. Clicking on the button opens a dialog that allows you to select the field and the order (‘Ascending’ or ‘Descending’ that the data is to be selected on.

On the next line on the left hand side we can select whether all the metrics are to be exported or only those metrics that are marked as active. On the right side we can select whether the data is exported in standard format (i.e. using the XML tags defined for each metric) or in JHawk Interchange format which can be used to recreate the metric record of a particular analysis set.

On the next line we can select the file name to which the data is to be exported. You can either enter a file name or use the browse button to select one.

## Creating XML files for the Data Viewer

The files used by the Data Viewer must be in the JHawk Metric Interchange Format (JMIF). These are XML files that have been created using the JMIF option when exporting files from JHawk either using the stand alone application, The JHawk Eclipse Plugin or the command line interface.

On the ‘Create XML’ tab you need to select the ‘Entire System’ and the ‘JHawk Interchange Format’ radio buttons. You then need to select the levels that you wish the analysis to occur at – Package, Class or Method. If you want to analyse to a particular level you will need to select the levels above so that the JHawk Data Viewer (which works on a tree basis) can ‘drill down’ to the lower levels. This means that if you want to see data down to the method level you will need to select the package, class and method levels and if you want to analyse to the class level you will need to select the package and class



levels. The screenshot below illustrates the choices that should be made to produce a JHawk Metric Interchange file at the method level -

The screenshot shows the 'Generate XML' dialog box in JHawk Data Viewer. The settings are as follows:

- Generate details for...**
  - ☐ All classes in selected package
  - ☐ All methods in selected class
  - ☒ Entire System
- Generate Details for each ..**
  - ☒ Package
  - ☒ Class
  - ☒ Method
- Select Sort Field for ...**
  - Packages
  - Classes..
  - Methods..
- Select Active/All Metrics**
  - ☒ Select All Metrics
  - ☐ Select Active Metrics only
- Output format**
  - ☐ Standard format
  - ☒ JHawk Interchange format
- XML File**: c:\MyXML
- Buttons**: Browse..., Create XML

The level to which you wish to carry out analysis will have a bearing on the size of the XML files created. This, in turn, may limit the number of files that JHawk Data Viewer can handle at a given level of memory. As an example the XML file sizes for an analysis of the entire source for Eclipse 3.4 (16,175 source files) were 215Mb at method level, 40Mb at class level and 134k at package level. You can analyse at class or package level but if you wish to analyse metrics that are ultimately based on data collected at method level (and this includes many of the important metrics such as the Halstead Metrics and Cyclomatic Complexity) you will need to analyse right down to the method level.

If you wish to compare files in the tree views (Text Compare and Graph) then all of your systems should have the same name (or all have no name). You can set the system name in the 'Select Files' tab on the 'System Details' tab. The default value of the System name is 'No System Name'.

### ***Creating files to load back into JHawk Stand Alone***

If you are creating files to view again in the JHawk stand alone version the same criteria apply as for the JHawk Data Viewer

# Output Formats

## CSV – Standard

In the standard CSV export format a summary record is written at each level in addition to the data at that level.

| LCM_NSPC.csv |  |                               |                            |                  |             |      |      |       |
|--------------|--|-------------------------------|----------------------------|------------------|-------------|------|------|-------|
|              | A  | B                             | C                          | D                | E           | F    | G    | H     |
| 1            | Details of classes for Package test.jhawk.csvexport.package2 |                               |                            |                  |             |      |      |       |
| 2            | Details of methods for Class ClassWithOneMethod              |                               |                            |                  |             |      |      |       |
| 3            | System   | Package                       | Name                       | COMP             | NOA         | NOCL | NOC  | VDEC  |
| 4            | No System Name   | test.jhawk.csvexport.package2 | theMethod                  | 1                | 1           | 0    | 0    | 1     |
| 5            | Class overview for class ClassWithOneMethod                  |                               |                            |                  |             |      |      |       |
| 6            | System   | Package                       | Name                       | Superclass       | No. Methods | LCOM | TCC  | MAXCC |
| 7            | No System Name   | test.jhawk.csvexport.package2 | ClassWithOneMethod         | java.lang.Object | 1           | 0    | 1    | 1     |
| 8            | Details of methods for Class ClassWithTwoMethods             |                               |                            |                  |             |      |      |       |
| 9            | System   | Package                       | Name                       | COMP             | NOA         | NOCL | NOC  | VDEC  |
| 10           | No System Name   | test.jhawk.csvexport.package2 | firstMethod                | 1                | 1           | 0    | 0    | 1     |
| 11           | No System Name   | test.jhawk.csvexport.package2 | secondMethod               | 1                | 2           | 0    | 0    | 1     |
| 12           | Class overview for class ClassWithTwoMethods                 |                               |                            |                  |             |      |      |       |
| 13           | System   | Package                       | Name                       | Superclass       | No. Methods | LCOM | TCC  | MAXCC |
| 14           | No System Name   | test.jhawk.csvexport.package2 | ClassWithTwoMethods        | java.lang.Object | 2           | 0    | 2    | 1     |
| 15           | Details of Methods for Package test.jhawk.csvexport.package2 |                               |                            |                  |             |      |      |       |
| 16           | System   | Package                       | Class                      | Name             | COMP        | NOA  | NOCL | NOC   |
| 17           | No System Name   | test.jhawk.csvexport.package2 | ClassWithTwoMethods        | firstMethod      | 1           | 1    | 0    | 0     |
| 18           | No System Name   | test.jhawk.csvexport.package2 | ClassWithTwoMethods        | secondMethod     | 1           | 2    | 0    | 0     |
| 19           | No System Name   | test.jhawk.csvexport.package2 | ClassWithOneMethod         | theMethod        | 1           | 1    | 0    | 0     |
| 20           | Details of classes for Package test.jhawk.csvexport.package1 |                               |                            |                  |             |      |      |       |
| 21           | Details of methods for Class ClassReferencingTwoClasses      |                               |                            |                  |             |      |      |       |
| 22           | System   | Package                       | Name                       | COMP             | NOA         | NOCL | NOC  | VDEC  |
| 23           | No System Name   | test.jhawk.csvexport.package1 | theMethod                  | 1                | 0           | 0    | 0    | 2     |
| 24           | Class overview for class ClassReferencingTwoClasses          |                               |                            |                  |             |      |      |       |
| 25           | System   | Package                       | Name                       | Superclass       | No. Methods | LCOM | TCC  | MAXCC |
| 26           | No System Name   | test.jhawk.csvexport.package1 | ClassReferencingTwoClasses | java.lang.Object | 1           | 0    | 1    | 1     |
| 27           | Details of Methods for Package test.jhawk.csvexport.package1 |                               |                            |                  |             |      |      |       |
| 28           | System   | Package                       | Class                      | Name             | COMP        | NOA  | NOCL | NOC   |
| 29           | No System Name   | test.jhawk.csvexport.package1 | ClassReferencingTwoClasses | theMethod        | 1           | 0    | 0    | 0     |
| 30           | System overview for No System Name                           |                               |                            |                  |             |      |      |       |
| 31           | Name   | No. Packages                  | No. Classes                | No. Methods      | NOS         | TCC  | AVCC | HBUG  |
| 32           | No System Name   | 2                             | 3                          | 4                | 16          | 3    | 0.75 | 0.11  |

## CSV – Raw

When data is exported in the raw CSV format a single record for each artefact is written at the requested level. In the example here the data has been exported at the method level. In each case the fully qualified name will be available with the System, package, class and method name written in the columns at the left.

| LPCM_NSPRawMethod.csv |                |                               |                            |              |      |     |      |     |      |      |     |      |     |      |      |       |
|-----------------------|----------------|-------------------------------|----------------------------|--------------|------|-----|------|-----|------|------|-----|------|-----|------|------|-------|
|                       | A              | B                             | C                          | D            | E    | F   | G    | H   | I    | J    | K   | L    | M   | N    | O    | P     |
| 1                     | System         | Package                       | Class                      | Name         | COMP | NOA | NOCL | NOC | VDEC | VREF | NOS | NEXP | MDN | HLTH | HVOC | HVOL  |
| 2                     | No System Name | test.jhawk.csvexport.package2 | ClassWithTwoMethods        | firstMethod  | 1    | 1   | 0    | 0   | 1    | 0    | 0   | 1    | 0   | 12   | 11   | 41.51 |
| 3                     | No System Name | test.jhawk.csvexport.package2 | ClassWithTwoMethods        | secondMethod | 1    | 2   | 0    | 0   | 1    | 0    | 0   | 1    | 0   | 14   | 12   | 50.19 |
| 4                     | No System Name | test.jhawk.csvexport.package1 | ClassReferencingTwoClasses | theMethod    | 1    | 0   | 0    | 0   | 2    | 0    | 0   | 2    | 0   | 17   | 13   | 62.91 |
| 5                     | No System Name | test.jhawk.csvexport.package2 | ClassWithOneMethod         | theMethod    | 1    | 1   | 0    | 0   | 1    | 0    | 0   | 1    | 0   | 12   | 11   | 41.51 |
| 6                     |                |                               |                            |              |      |     |      |     |      |      |     |      |     |      |      |       |
| 7                     |                |                               |                            |              |      |     |      |     |      |      |     |      |     |      |      |       |
| 8                     |                |                               |                            |              |      |     |      |     |      |      |     |      |     |      |      |       |

## XML – Standard

XML output can be written in two forms – Standard or JHawk Interchange . In the standard format xml output is written for the selected metrics (active or all) and at the relevant levels (System, Package, Class, Method etc). Sample output is shown below.

```
<?xml version="1.0" ?>
- <System>
  <Name>No System Name</Name>
  <Time>1266958329796</Time>
  <Locale>en_IE</Locale>
- <Packages>
  - <Package>
    <OwningSystem>No System Name</OwningSystem>
    <Name>test.jhawk.csvexport.package2</Name>
  - <Metrics>
    <name>test.jhawk.csvexport.package2</name>
    <numberOfClasses>2</numberOfClasses>
    <numberOfMethods>3</numberOfMethods>
    <numberOfStatements>10</numberOfStatements>
    <tcc>2</tcc>
    <avcc>0.6666666666666666</avcc>
    <halsteadCumulativeBugs>0.06658274081814274</halsteadCumulativeBugs>
    <halsteadEffort>579.0806916281736</halsteadEffort>
    <halsteadCumulativeLength>64</halsteadCumulativeLength>
    <halsteadCumulativeVolume>199.7482224544282</halsteadCumulativeVolume>
    <maintainabilityIndex>135.2968608614708</maintainabilityIndex>
    <maintainabilityIndexNC>135.2968608614708</maintainabilityIndexNC>
    <abstractness>0.0</abstractness>
    <fanin>0</fanin>
    <fanout>0</fanout>
    <instability>0.0</instability>
    <distance>1.0</distance>
    <maxcc>1</maxcc>
    <cumulativeNumberOfComments>0</cumulativeNumberOfComments>
    <cumulativeNumberOfCommentLines>0</cumulativeNumberOfCommentLines>
    <loc>15</loc>
  </Metrics>
- <Classes>
  - <Class>
    <OwningPackage>test.jhawk.csvexport.package2</OwningPackage>
    <ClassName>ClassWithOneMethod</ClassName>
  - <Metrics>
    <name>ClassWithOneMethod</name>
    <superclass>java.lang.Object</superclass>
    <numberOfMethods>1</numberOfMethods>
    <lcom>0.0</lcom>
    <tcc>1</tcc>
    <maxcc>1</maxcc>
    <avcc>1.0</avcc>
    <numberOfStatements>3</numberOfStatements>
    <halsteadCumulativeBugs>0.016504393141215858</halsteadCumulativeBugs>
    <halsteadEffort>133.08010665230543</halsteadEffort>
```

---

## XML – JHawk Interchange Format

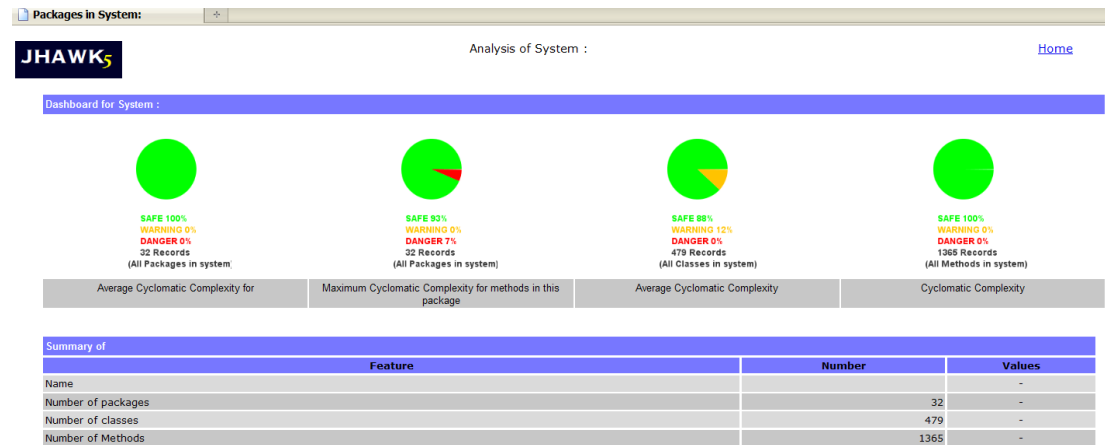
If you select the JHawk Interchange format for your XML output then the data is exported for the entire system at all levels. As this is the base data collected for your system this data can be used to completely rebuild the dataset inside either JHawk or the JHawk DataViewer add on.

```
<?xml version="1.0" ?>
<SYS>
  <NAM>No System Name</NAM>
  <TIM>1266958351640</TIM>
  <LLE>en_IE</LLE>
- <PKCS>
- <PCK>
  <OWS>No System Name</OWS>
  <NAM>test.jhawk.csvexport.package2</NAM>
  <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
- <CLSS>
- <CLS>
  <OWP>test.jhawk.csvexport.package2</OWP>
  <CNM>ClassWithOneMethod</CNM>
- <MODS>
  <MOD>public</MOD>
</MODS>
  <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
- <MTHS>
- <MTH>
  <CNM>test.jhawk.csvexport.package2.ClassWithOneMethod</CNM>
  <NAM>theMethod</NAM>
  <RET>void</RET>
- <ARGS>
  <ARG K="aParameter" V="java.lang.String" />
</ARGS>
- <CLRS>
  <CLR K="java.lang.String" V="2" />
</CLRS>
  <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
  <BAS LB="1" MN="0" CW="0" EW="0" EX="1" LO="0" ND="7" NR="5" UD="6" UR="5" ST="2" TN="0" LC="3" />
- <VDES>
  <VDE K="temp" V="java.lang.String" />
</VDES>
- <MODS>
  <MOD>public</MOD>
</MODS>
</MTH>
</MTHS>
  <SCL>java.lang.Object</SCL>
</CLS>
- <CLS>
  <OWP>test.jhawk.csvexport.package2</OWP>
  <CNM>ClassWithTwoMethods</CNM>
- <MODS>
  <MOD>public</MOD>
</MODS>
```

---

## HTML

HTML data is output at the levels selected and the structure of the pages output will depend on the levels selected. At the very least a root page will be output showing the highest level select. The data from all the other levels selected can be accessed from this.



If you have elected to select all levels then the first page of the HTML output will be as above. All the active dashboard gauges will be displayed and a table setting out the system level metrics will be displayed.

Below this a list of all the packages that constitute the system being analysed will be displayed.

Packages in System: +

Lines Of Code 5675 -

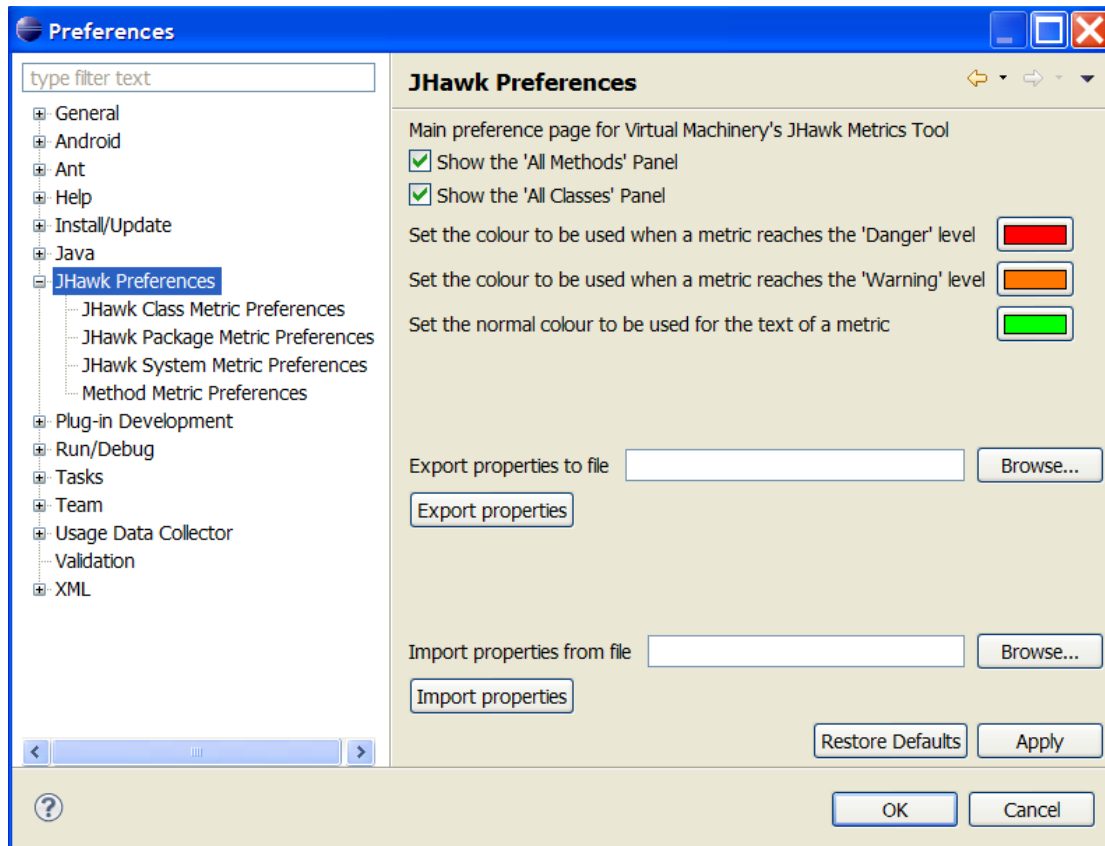
| Name                                   | No. Classes | No. Methods | NOS | TCC | AVCC | HBUG | HEFF     | HLTH | HVOL     | MI     | MINC   | ABST | FIN | FOUT | INST | DIST | MAXCC | CCOM | CCML | NLOC |
|--|-------------|-------------|-----|-----|------|------|----------|------|----------|--------|--------|------|-----|------|------|------|-------|------|------|------|
| <a href="#">junit.samples</a>          | 3           | 16          | 64  | 4   | 0.25 | 0.61 | 11983.03 | 445  | 1834.75  | 166.11 | 126.23 | 0.00 | 0   | 3    | 1.00 | 0.00 | 2     | 7    | 34   | 80   |
| <a href="#">junit.samples.money</a>    | 4           | 63          | 208 | 13  | 0.21 | 2.39 | 47566.93 | 1700 | 7179.81  | 180.90 | 130.90 | 0.25 | 0   | 1    | 1.00 | 0.25 | 7     | 28   | 69   | 283  |
| <a href="#">junit.tests</a>            | 2           | 3           | 10  | 2   | 0.67 | 0.13 | 3498.80  | 94   | 389.47   | 169.03 | 129.15 | 0.00 | 0   | 3    | 1.00 | 0.00 | 1     | 2    | 8    | 13   |
| <a href="#">junit.tests.extensions</a> | 12          | 25          | 119 | 14  | 0.56 | 1.09 | 19253.62 | 778  | 3277.80  | 125.94 | 125.94 | 0.00 | 0   | 2    | 1.00 | 0.00 | 2     | 5    | 15   | 144  |
| <a href="#">junit.tests.framework</a>  | 37          | 129         | 436 | 41  | 0.32 | 3.77 | 60100.74 | 2817 | 11298.91 | 132.45 | 132.45 | 0.00 | 0   | 2    | 1.00 | 0.00 | 2     | 37   | 72   | 618  |
| <a href="#">junit.tests.runner</a>     | 19          | 39          | 153 | 22  | 0.56 | 1.69 | 37204.05 | 1137 | 5083.69  | 129.87 | 129.87 | 0.00 | 0   | 1    | 1.00 | 0.00 | 3     | 7    | 13   | 192  |
| <a href="#">org.hamcrest</a>           | 7           | 50          | 147 | 16  | 0.32 | 2.08 | 52417.52 | 1462 | 6239.30  | 156.86 | 136.25 | 0.71 | 0   | 3    | 1.00 | 0.71 | 3     | 41   | 171  | 199  |
| <a href="#">org.hamcrest.core</a>      | 10          | 55          | 138 | 22  | 0.40 | 1.74 | 41706.27 | 1278 | 5218.81  | 177.19 | 137.30 | 0.00 | 0   | 1    | 1.00 | 0.00 | 4     | 31   | 124  | 218  |
| <a href="#">org.hamcrest.internal</a>  | 3           | 10          | 22  | 4   | 0.40 | 0.18 | 2514.03  | 146  | 531.02   | 141.08 | 141.08 | 0.00 | 0   | 2    | 1.00 | 0.00 | 2     | 0    | 0    | 33   |
| <a href="#">org.junit.samples</a>      | 2           | 14          | 60  | 3   | 0.21 | 0.67 | 13747.92 | 441  | 1821.47  | 126.23 | 126.23 | 0.00 | 0   | 0    | 0.00 | 1.00 | 2     | 4    | 10   | 68   |

This is presented in tabular form with a column for each of the metrics calculated at this level (remember that you can choose whether your HTML output shows all metrics or only those that you have marked as active). There is a row for each package and if any metric in the row has a valid range, and lies outside that range then either a red danger or an orange warning indicator will be displayed to the left of the row. The background of the offending metric(s) will also be set to either red or orange.

You can click on the link in the name column to drill down to the next level. This may or may not be active depending on the levels that you have selected to be displayed in the table.

All pages are basically laid out the same – one or more lines of dashboard gauges (if these have been activated) followed by a list of metrics at that particular level. Below this will be a table of data relating to the level below this with links to that level.

# The Preferences Tab

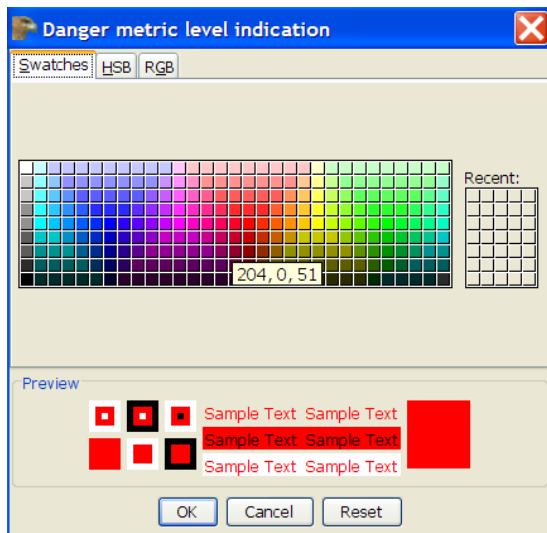


The JHawk Preferences are accessed via the Eclipse 'Window' menu – then select 'Preferences', then 'JHawk Preferences'. The dialog tab opens on the 'General' preferences.. The 'System', 'Package', 'Class' and 'Method' sub-tabs allow you to configure the metrics available at each of the System, Package, Class and Method levels.

## General Preferences

The General Preferences tab allows you to set the following attributes of JHawk.

- Whether the 'All Methods' panel (a sub-tab of the Results panel) will be populated and displayed. This is left optional as populating this panel can substantially increase the memory and processing burden on JHawk if very large numbers of methods are being analysed.
- Whether the 'All Classes' panel (a sub-tab of the Results panel) will be populated and displayed. This is left optional as populating this panel can substantially increase the memory and processing burden on JHawk if very large numbers of classes are being analysed.
- Setting the colours used to indicate the Normal, Warning and Danger levels of metrics in the metrics tables and on the dashboard. When the button is pressed the following dialog pops up and the colour to be used can be selected –



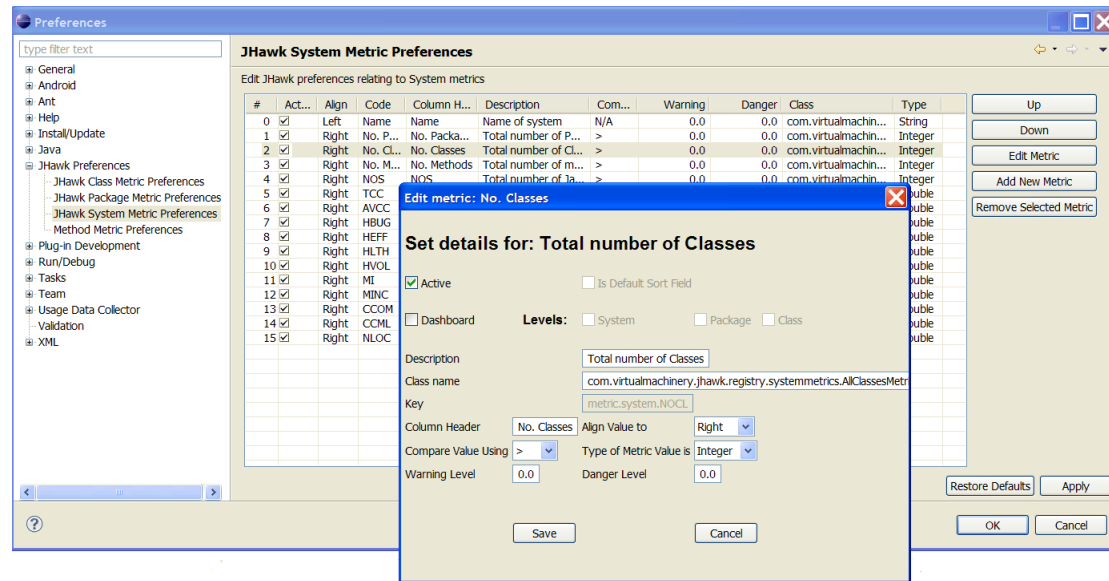
## ***Importing and exporting settings***

You may have created a set of preferences that you wish to keep for later use (for example you may want to use different preferences for different Java code sets). You do this using this screen. Having created your metrics using the preceding three tabs and stored them using the ‘Store’ button on each screen you can export them to a file. Then if you wish to re-use these metrics you can load them up from the file in one of four ways –

1. Using this tab select the file by using the ‘Browse’ button beside the ‘Import Properties from file’ text entry. Then press the ‘Import’ button – this will load up the metrics and apply them to your currently selected file set. This may take some time if you have a very large file set.
2. On the ‘Welcome’ tab select the ‘Use a previously saved set of preferences’ ‘Browse’ button and select the file that you wish to load – then press the graphic button the left hand side of the line - this will load up the metrics and apply them to any file set that you have already selected. This may take some time if you have a very large file set. If you have not yet selected a file set then the preferences will be applied to the files that you select
3. When you call JHawk using the command line interface using the `-p` option e.g. `-p mynewproperties.properties`
4. Rename the file to `jhawk.properties` – by default JHawk will always start up with the `jhawk.properties` file.

If your properties file is corrupt for any reason JHawk may not start. In this case you should use the last known good properties file. If you cannot find a non-corrupt file then you can go back to the file from the original distribution.

## Metrics sub-tabs – Edit Metric Column details



This tab allows you to amend the details of existing metrics and to add and delete metrics. It also allows you to set the order in which metrics will appear in the JHawk tables and which metric will be used as the default sort field. There is a separate tab for each level of metrics – System, Package, Class, Method.

The details relating to the metrics are stored in the `jhawk.properties` file. All changes to metrics need to be made via the metrics preferences screen. Attempting to edit the `jhawk.properties` file will probably result in JHawk failing to start successfully. If this happens, or you believe the `jhawk.properties` file to be corrupt you should replace the file with the original `jhawk.properties` file that came with your distribution of JHawk. The same format is used for the `jhawk.properties` file across all versions of the JHawk product – the Stand alone version, the Eclipse plugin, the command line and the Data viewer.

To save the file after editing it press the apply button – this will cause your changes to be reflected in the JHawk Metrics View.

To export the current settings use the main JHawk Preferences page.

To import a previously saved set of settings use the main JHawk Preferences page.

## The Buttons

- The Up button – this moves a metric up the order that it will appear in the metric tables. Moving the metric ‘up’ the order means that it will display further to the left in the table.
- The Down button – this moves a metric down the order that it will appear in the metric tables. Moving the metric ‘up’ the order means that it will display further to the right in the table.
- Edit Metric – this opens a dialog which allows you to edit the currently selected metric (see diagram above and separate section below)
- Add New metric – this allows you to add a new metric – see the separate section on this below.
- Remove Selected Metric – this will delete the selected metric from the list of metrics. Unless this is a metric that you have added then you should consider very carefully whether you want to do this. If all you want to do is remove a metric from the tables displayed then you should inactivate it using the ‘Active’ check box in the metric editing screen – see below.



## Add a metric and edit details of an existing metric

When you are editing a metric it is most likely that you will want to change the following information about a particular metric –

**Active** – use this to control whether a metric is displayed in any of the tables.

**On Dashboard** – Enable this to indicate that you want a metric to appear on the JHawk dashboard panel. There are three level indicators which allow you to indicate which level(s) you want the metric to be assessed at. The available levels will depend on the level of the metric selected.

**Description** – You might choose to describe the metric in your own language or in a different way

**Column Header** – Again you might change the header to be more descriptive.

**Warning Level** – You can set the level at which the metric will displayed in the warning colour in the tables and on the dashboard pie charts. This and the danger level will only be available for numerical and list metrics. In the case of list metrics the value will be compared against the count of the number of entries in the list.

**Danger Level** - You can set the level at which the metric will displayed in the danger colour in the tables and on the dashboard pie charts.

It is unlikely that you will have any need to change the following attributes of a metric except when you are adding a new metric –

**Align Value** – This is a dropdown selection box that allows you to set the alignment of the metric data in a table or display field. The options are LEFT, RIGHT or CENTER

**Compare value using:** - This value sets the direction of comparison against the Warning and Danger Levels this can either be > (Greater than – the metric will be marked as being in the Warning or Danger zone if the value of the metric is greater than the levels), < (Less than – the metric will be marked as being in the Warning or Danger zone if the value of the metric is greater than the levels) or N/A (Not applicable)

**Metric type** – This signifies the metric type – the available options are INTEGER, DOUBLE, STRING and LIST. A LIST metric will also provide a count of the values in the list for the purposes of the warning/danger levels

You should only delete metrics that you have created yourself. If all that you want to do is prevent a metric being displayed in the listings then all you have to do is to change the Active flag – see below.

The screenshot shows the 'JHawk System Metric Preferences' dialog box. A sub-dialog titled 'Add System metric' is open, showing the 'Set details for:' section. The sub-dialog has a close button (X) in the top right corner. The 'Set details for:' section contains the following fields and options:

- ☐ Active
- ☐ Is Default Sort Field
- ☐ Dashboard
- Levels:** ☐ System ☐ Package ☐ Class
- Description:
- Class name:
- Key:
- Column Header:  Align Value to:
- Compare Value Using:  Type of Metric Value is:
- Warning Level:  Danger Level:
-

When you add a new metric you will get virtually the same dialog but you will see there a number of new fields available to you when you come to add a metric. You can now set the class name and key fields. You need to be careful setting these as they are vital to the working of JHawk.

In the case of the class name you will need to provide the full name (i.e. including package name) of the class. The class will also have to be visible to JHawk (you can find out how to do this in the section on creating a new metric). Before allowing you to save a new Metric entry to the preferences store, JHawk will check that the class can be instantiated and will display an error if this is unsuccessful.

The metric key must start with “metric.” and then be followed by either “system.”, “package.”, “class.” or “method.” - this value being dependant on the level at which the metric is being added (as shown in the title of the dialog). The key at the end must be unique to that particular level. Before allowing you to save a new Metric entry to the preferences store, JHawk will check that the key is valid and will display an error if this is not the case.