The JFreeChart Class Library

Version 0.9.8

Developer Guide

Written by David Gilbert

April 25, 2003

Contents

1 Introduction

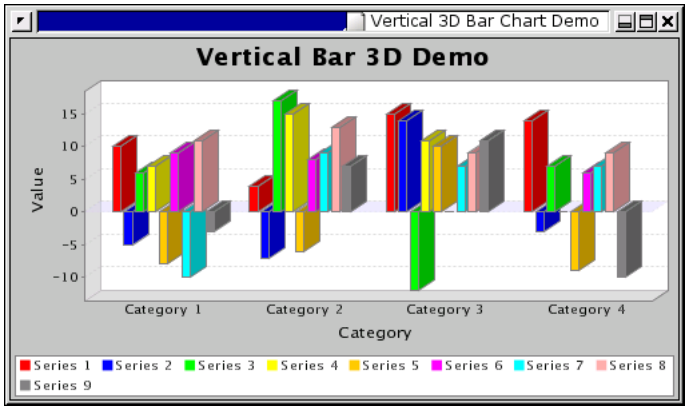
1.1 What is JFreeChart?

1.1.1 Overview

JFreeChart is a free Java chart library. It is distributed with complete source code, subject to the terms of the GNU Lesser General Public Licence (see Appendix A for details).

1.1.2 Features

JFreeChart can generate pie charts, bar charts (horizontal and vertical, regular and stacked, optional 3D-effect), line charts, scatter plots, time series charts (including moving averages, high-low-open-close charts and candlestick plots), Gantt charts, meter charts (dial, compass and thermometer), symbol charts, wind plots, combination charts and more.



Additional features include:

- tool tips;
- interactive zooming;
- chart mouse events;
- annotations;
- data is accessible from any implementation of the defined interfaces;
- export to JPEG, PNG, SVG, PDF and any other format with a `Graphics2D` implementation;
- HTML image map generation;
- works in applications, servlets, JSP (thanks to the Cewolf project[1]) and applets;

[1] See `http://cewolf.sourceforge.net` for details.

- distributed with complete source code subject to the terms of the GNU Lesser General Public License (LGPL);

JFreeChart is written entirely in Java, and should run on any implementation of the Java 2 platform (JDK1.3 or later recommended).

1.1.3 Home Page

The JFreeChart home page can be found at:

`http://www.jfree.org/jfreechart/index.html`

Here you will find all the latest information about JFreeChart, including sample charts, download links, Javadocs, a support forum and more.

1.2 This Document

1.2.1 Versions

Two versions of this document are available:

- a free version, the "JFreeChart Installation Guide", is available from the JFreeChart home page, and contains chapters up to and including the instructions for installing JFreeChart.
- a premium version, the "JFreeChart Developer Guide", is available for purchase from the *Kagi internet store* (see the link on the JFreeChart home page), and includes additional tutorial chapters and reference documentation for the JFreeChart classes.

Proceeds from the sale of the JFreeChart Developer Guide are used to sponsor on-going development of JFreeChart.

1.2.2 Disclaimer

Please note that I have put in considerable effort to ensure that the information in this document is up-to-date and accurate, but I cannot guarantee that it does not contain errors. You must use this document *at your own risk* or *not use it at all.*

1.3 Acknowledgements

JFreeChart contains code and ideas from many people. At the risk of missing someone out, I would like to thank the following people for contributing to the project: Richard Atkinson, David Berry, Anthony Boulestreau, Jeremy Bowman, Søren Caspersen, Chuanhao Chiu, Pascal Collet, Martin Cordova, Paolo Cova, Michael Duffy, Jonathan Gabbai, Serge V. Grachov, Hans-Jurgen Greiner, Joao Guilherme Del Valle, Aiman Han, Jon Iles, Wolfgang Irler, Xun Kang, Bill Kelemen, Norbert Kiesel, Gideon Krause, David Li, Tin Luu, Craig MacFarlane, Achilleus Mantzios, Thomas Meier, Jim Moore, Jonathan Nash, David M. O'Donnell, Krzysztof Paz, Tomer Peretz, Andrzej Porebski, Viktor Rajewski, Michael Rauch, Cameron Riley, Dan Rivett, Thierry Saura, Andreas Schneider, Jean-Luc Schwab, Bryan Scott, Roger Studner, Irv Thomae, Eric Thomas,

Rich Unger, Daniel van Enckevort, Laurence Vanhelsuwé, Sylvain Vieujot, Mark Watson, Alex Weber, Matthew Wright, Christian W. Zuckschwerdt, Hari and Sam (oldman).

1.4 Comments and Suggestions

If you have any comments or suggestions regarding this document, please send e-mail to: `david.gilbert@object-refinery.com`

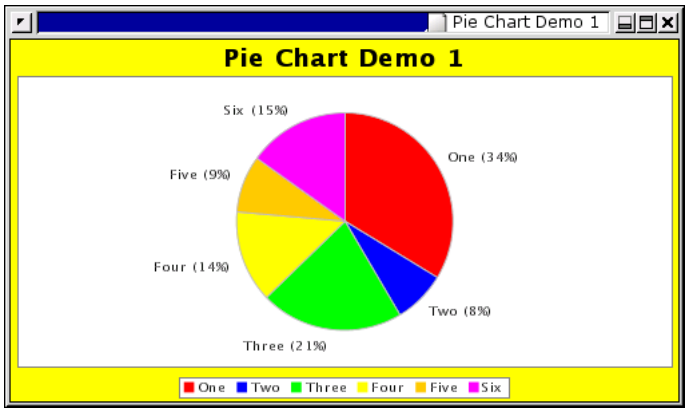
2 Sample Charts

2.1 Introduction

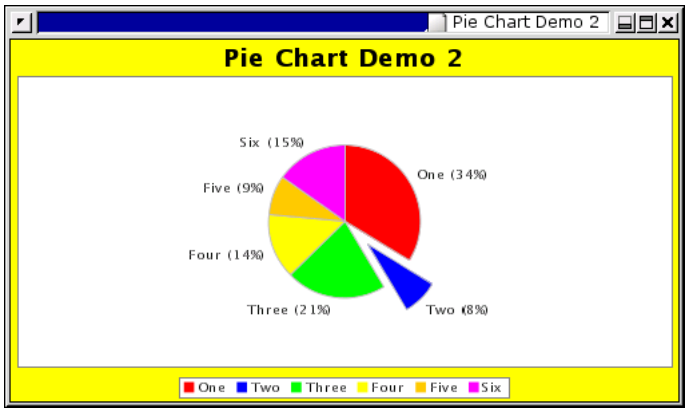
This section shows some sample charts created using JFreeChart. It is intended to give a reasonable overview of the types of charts that JFreeChart can generate.

2.2 Pie Charts

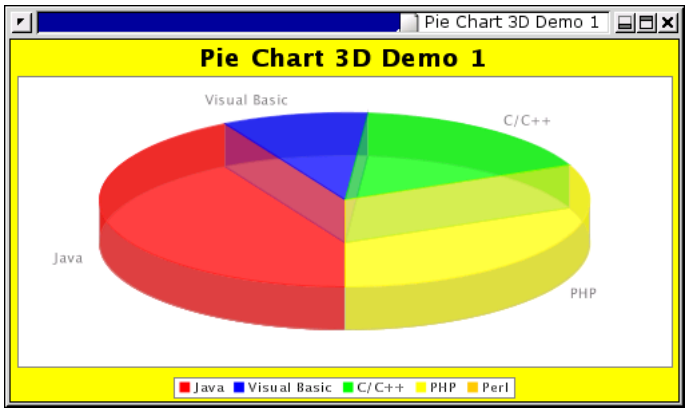
JFreeChart can create *pie charts* using any data that conforms to the `PieDataset` interface:



Individual pie sections can be "exploded":



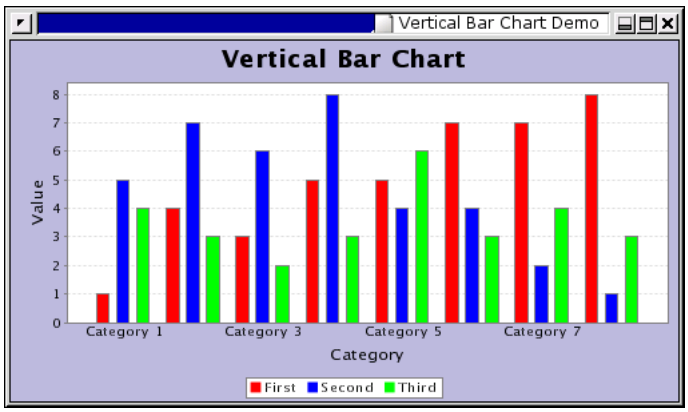
You can also display pie charts with a 3D effect:



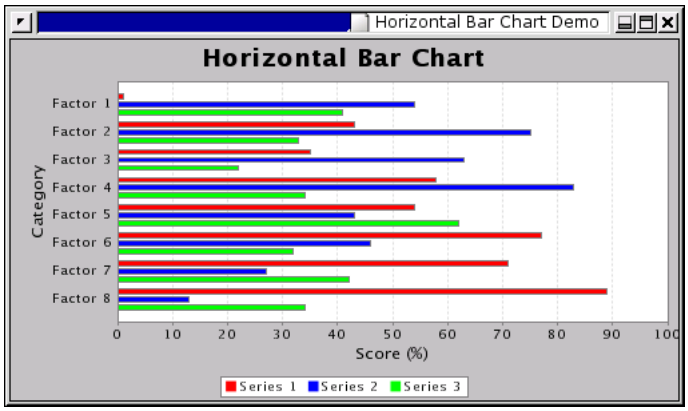
2.3 Bar Charts

A range of bar charts can be created with JFreeChart, using any data that conforms to the `CategoryDataset` interface.

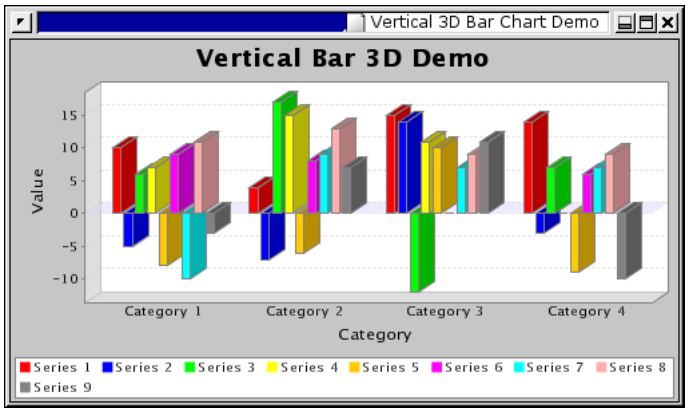
The first example is a *vertical bar chart*:



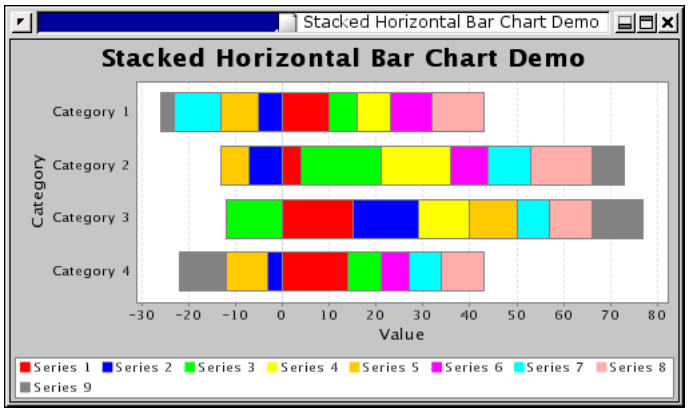
Changing the orientation, but still using a `CategoryDataset`, JFreeChart can generate a *horizontal bar chart*:



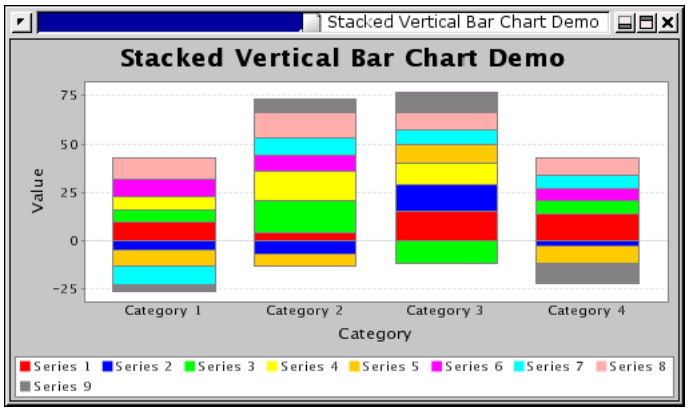
Both the vertical and horizontal bar charts can be displayed with a 3D effect:



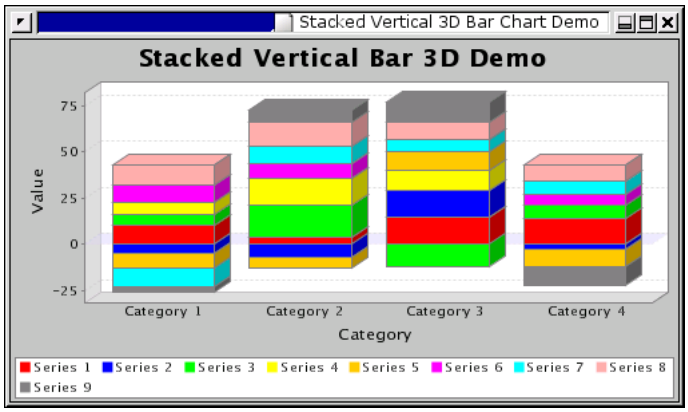
The bars can be stacked in a *stacked horizontal bar chart*:



...and similarly a *stacked vertical bar chart*:

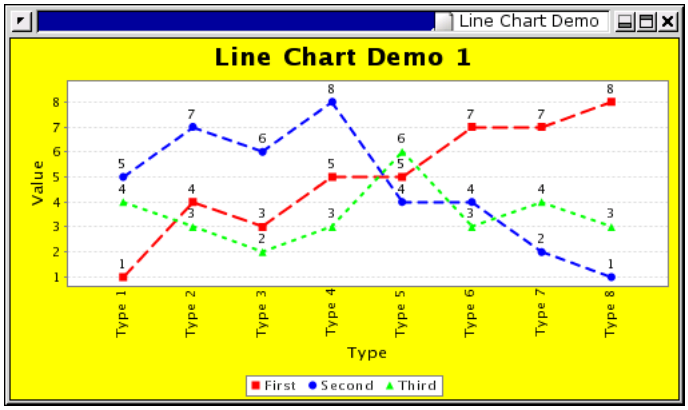


The stacked vertical bar chart can be displayed with a 3D effect:



2.4 Line Chart

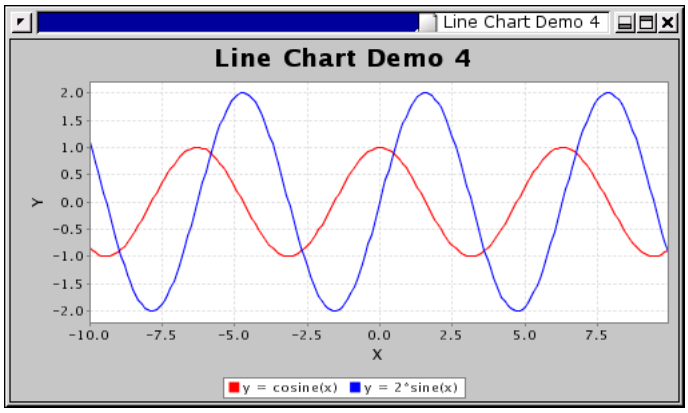
The *line chart* is generated using the same `CategoryDataset` that is used for the bar charts:



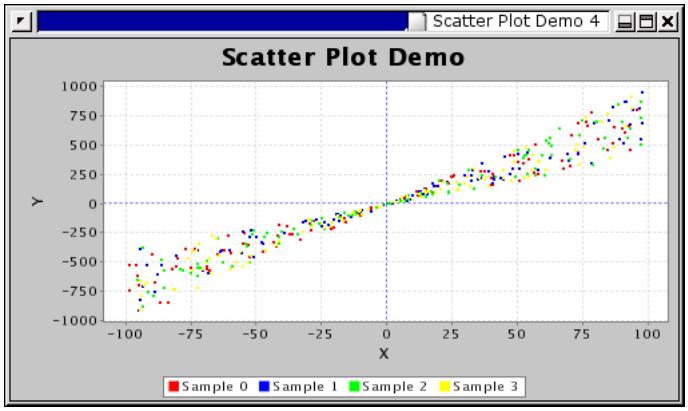
The data is the same, but the *line chart* gives you another presentation option.

2.5 XY Plots

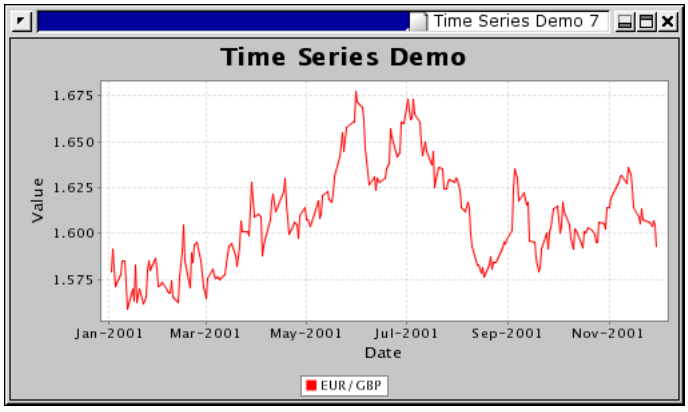
A third type of dataset, the `XYDataset`, is used to generate further chart types. The standard *XY plot* has numerical x and y axes. By default, lines are drawn between each data point:



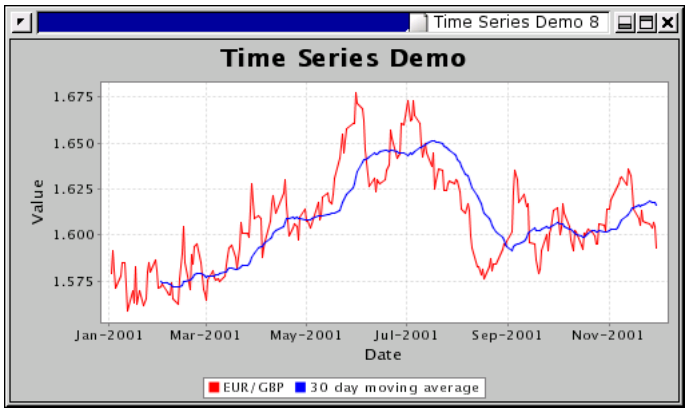
Dots can be drawn at data points, rather than connecting points with lines, for a *scatter plot*:



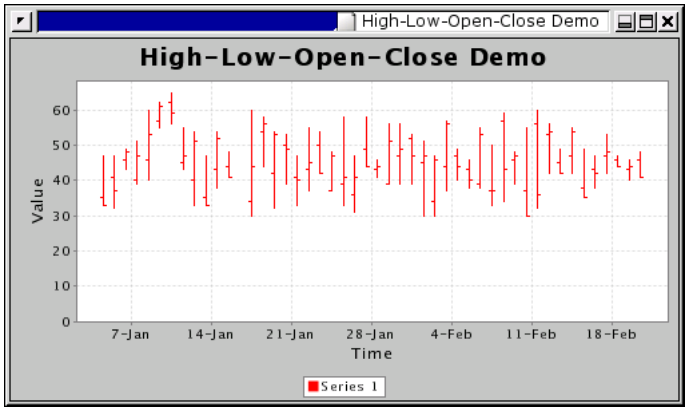
JFreeChart supports *time series charts*:



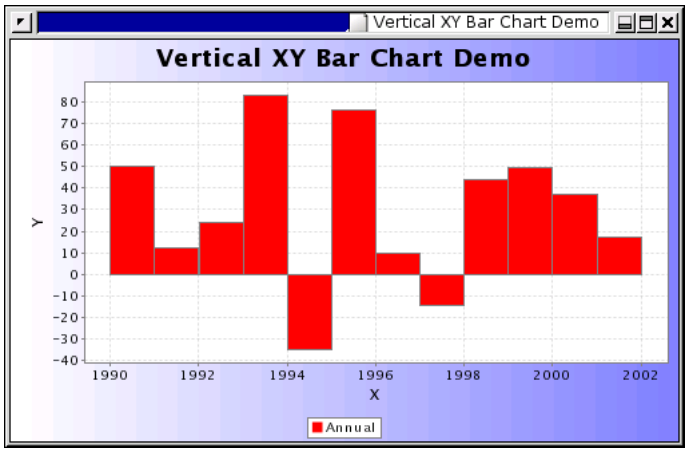
It is straightforward to add a moving average line to a time series chart:



Using a `HighLowDataset` (an extension of `XYDataset`) you can display *high-low-open-close* data:

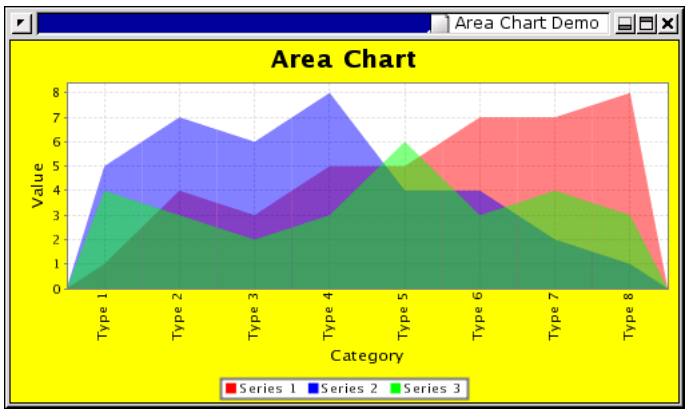


Using an `IntervalXYDataset` (another extension of `XYDataset`), JFreeChart can produce bar charts over a numerical domain:



2.6 Area Charts

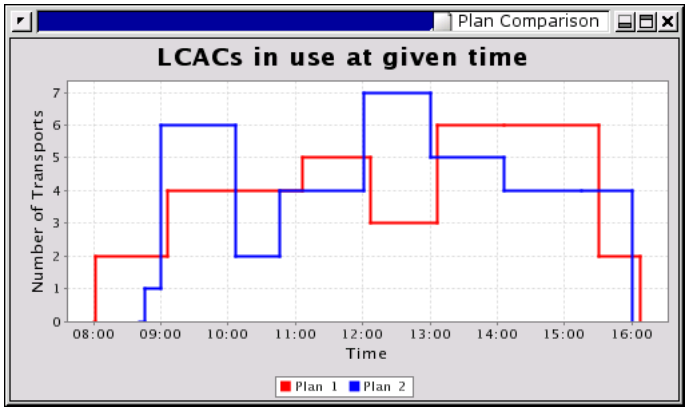
You can generate an *area chart* for data in a `CategoryDataset` or an `XYDataset`. For example:



JFreeChart also supports the creation of *stacked area charts*.

2.7 Step Chart

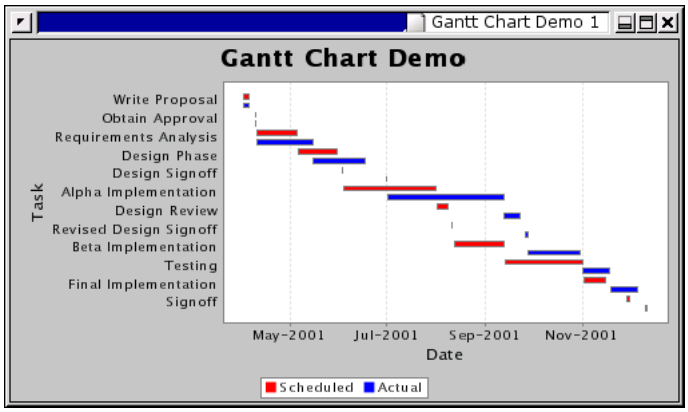
A *step chart* displays numerical data as a sequence of "steps":



Step charts are generated from data in an `XYDataset`.

2.8 Gantt Chart

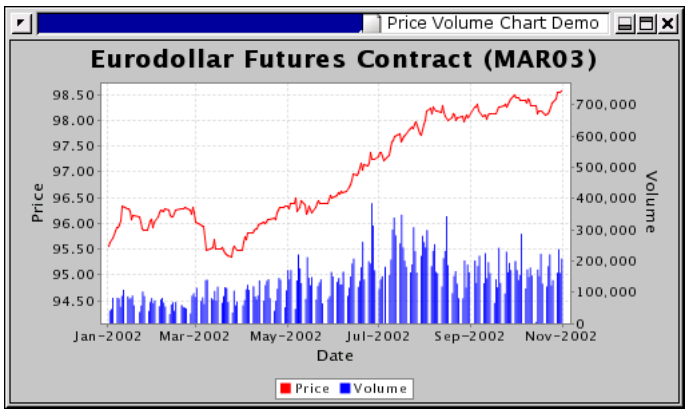
Gantt charts can be generated using data from an `IntervalCategoryDataset`:



From 0.9.5 onwards, it is possible to display multiple sub-periods within a single task.

2.9 Dual Axis Charts

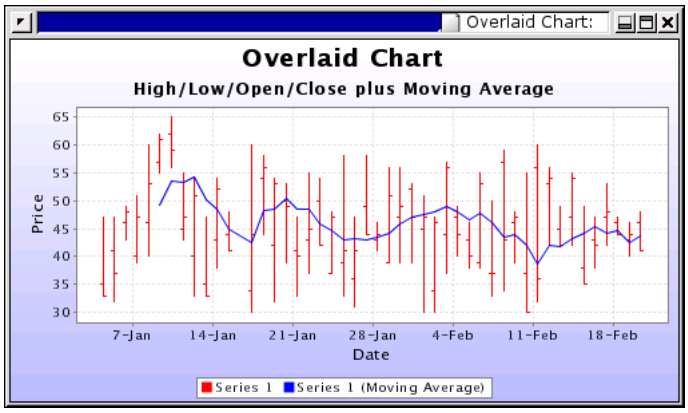
Charts with two range axes can be created with JFreeChart (from version 0.9.5):



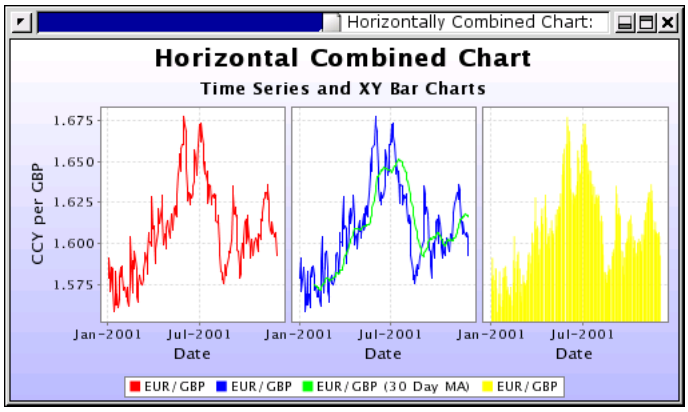
This feature is supported by the `CategoryPlot` class and the `XYPlot` class.

2.10 Combined Charts

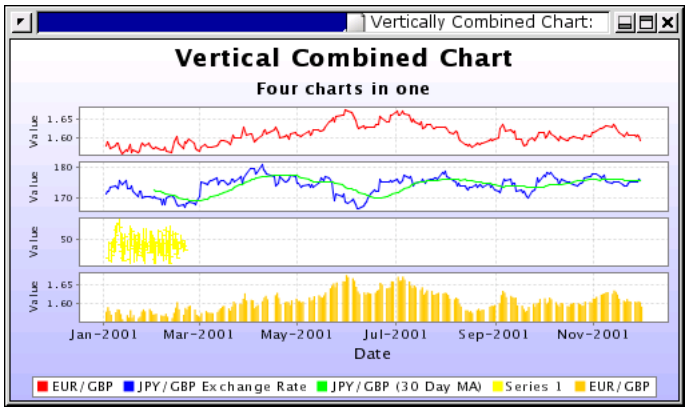
JFreeChart supports combined charts, including *overlaid charts*:



...*horizontally combined* charts:



...and *vertically combined* charts:



2.11 Future Development

JFreeChart is *free software*,[2] so anyone can extend it and add new features to it. Already, more than 50 developers from around the world have contributed

[2] See `http://www.fsf.org`

code back to the JFreeChart project. It is likely that many more chart types will be developed in the future as developers modify JFreeChart to meet their requirements. Check the JFreeChart home page regularly for announcements and other updates:

`http://www.jfree.org/jfreechart/index.html`

And if you would like to contribute code to the project, please join in...

3 Downloading and Installing JFreeChart

3.1 Introduction

This section contains instructions for downloading, unpacking, and (optionally) recompiling JFreeChart. Also included are instructions for running the JFreeChart demonstration application, and generating the Javadoc HTML files from the JFreeChart source code.

3.2 Download

You can download the latest version of JFreeChart from:

```
http://www.jfree.org/jfreechart/index.html
```

There are two versions of the JFreeChart download:

File:	Description:
`jfreechart-0.9.8.tar.gz`	JFreeChart for Linux/Unix.
`jfreechart-0.9.8.zip`	JFreeChart for Windows.

The two files contain the same source code. The main difference is that all the text files in the `zip` download have been recoded to have both carriage return *and* line-feed characters at the end of each line.

JFreeChart uses the JCommon class library (currently version `0.7.4`). The JCommon runtime jar file is included in the JFreeChart download, but if you require the source code (recommended) then you should also download JCommon from:

```
http://www.jfree.org/jcommon/index.html
```

There is a separate PDF document for JCommon, which includes full instructions for downloading and unpacking the files.

3.3 Unpacking the Files

After downloading JFreeChart, you need to unpack the files. You should move the download file to a convenient directory—when you unpack JFreeChart, a new subdirectory (`jfreechart-0.9.8`) will be created in the same location as the download file.

3.3.1 Unpacking on Linux/Unix

To extract the files from the download on Linux/Unix, enter the following command:

```
tar xvzf jfreechart-0.9.8.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.8`.

3.3.2 Unpacking on Windows

To extract the files from the download on Windows, enter the following command:

```
jar -xvf jfreechart-0.9.8.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.8`.

3.3.3 The Files

The top-level directory (`jfreechart-0.9.8`) contains the files and directories listed in the following table:

File/Directory:	Description:
`ant`	A directory containing an Ant `build.xml` script.
`checkstyle`	A directory containing a Checkstyle property file. This defines the coding conventions used in the JFreeChart source code.
`jfreechart-0.9.8.jar`	The JFreeChart runtime jar file.
`jfreechart-0.9.8-demo.jar`	A jar file containing demo applications.
`junit`	A directory containing JUnit testing code.
`lib`	A directory containing libraries used by JFreeChart.
`licence-LGPL.txt`	The GNU LGPL.
`README`	Important information - *read this first!*
`src`	A directory containing the source code for JFreeChart.

You should spend some time familiarising yourself with the files included in the download. In particular, you should always read the `README` file.

3.4 Running the Demonstration Applications

A range of demonstration applications are included with JFreeChart, to give you some idea of what the class library can do. It is not necessary to recompile the library to run the demonstration applications. All the classes are precompiled in the jar files.

To run the main demo (`JFreeChartDemo`), type the following command:

```
java -jar jfreechart-0.9.8-demo.jar
```

Alternatively, you can specify the classpath manually:

```
java -classpath lib/jcommon-0.7.4.jar:jfreechart-0.9.8.jar:
jfreechart-0.9.8-demo.jar org.jfree.chart.demo.JFreeChartDemo
```

Windows users should use a semi-colon rather than a colon to separate items on the classpath.

3.5 Compiling the Source

To recompile the JFreeChart classes, you can use the Ant `build.xml` file included in the distribution. Change to the `ant` directory and type:

```
ant compile
```

This will recompile all the necessary source files and recreate the JFreeChart run-time jar file.

To run the script requires that you have Ant 1.5.1 (or later) installed on your system, to find out more about Ant visit:

```
http://ant.apache.org/
```

3.6 Generating the Javadoc Documentation

The JFreeChart source code contains extensive *Javadoc comments*. You can use the `javadoc` tool to generate HTML documentation files directly from the source code—there is a link to the Javadoc HTML pages on the JFreeChart web page.

To generate the documentation, use the `javadoc` target in the Ant `build.xml` script:

```
ant javadoc
```

This will create a `javadoc` directory containing all the Javadoc HTML files, inside the main `jfreechart-0.9.8` directory.

4 Using JFreeChart

4.1 Overview

This section presents a simple introduction to JFreeChart, intended for new users of JFreeChart.

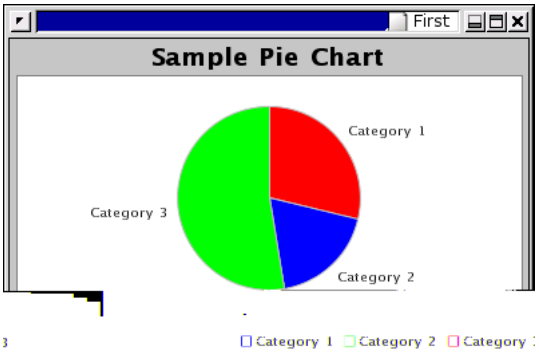
4.2 Creating Your First Chart

4.2.1 Overview

Creating charts with JFreeChart is a three step process. You need to:

- create a dataset containing the data to be displayed in the chart;
- create a `JFreeChart` object that will be responsible for drawing the chart;
- draw the chart to some output target (often, but not always, a panel on the screen);

To illustrate the process, we describe a sample application (`First.java`, included in the JFreeChart distribution) that produces this pie chart:



Each of the three steps outlined above is described, along with sample code, in the following sections.

4.2.2 The Data

Step one requires us to create a dataset for our chart. This can be done easily using the `DefaultPieDataset` class, as follows:

```
// create a dataset...
DefaultPieDataset data = new DefaultPieDataset();
data.setValue("Category 1", 43.2);
data.setValue("Category 2", 27.9);
data.setValue("Category 3", 79.5);
```

Note that JFreeChart can create pie charts using data from *any* class that implements the `PieDataset` interface. The `DefaultPieDataset` class (used above) provides a convenient implementation of this interface, but you are free to develop an alternative dataset implementation if you want to.[3]

[3] This is similar in concept to the way that Swing's `JTable` class obtains data via the `TableModel` interface. In fact, this was the inspiration for using interfaces to define the datasets for JFreeChart.

4.2.3 Creating a Pie Chart

Step two concerns how we will present the dataset created in step one. We need to create a `JFreeChart` object that can draw a chart using the data from our pie dataset. We will use the `ChartFactory` class, as follows:

```
// create a chart...
JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart",
                                               data,
                                               true,    // legend?
                                               true,    // tooltips?
                                               false);  // URLs?
```

Notice how we have passed a reference to the dataset to the factory method. JFreeChart keeps a reference to this dataset so that it can obtain data later on when it is drawing the chart.

The chart that we have created uses default settings for most attributes. There are many ways to customise the appearance of charts created with JFreeChart, but in this example we will just accept the defaults.

4.2.4 Displaying the Chart

The final step is to display the chart somewhere. JFreeChart is very flexible about where it draws charts, thanks to its use of the `Graphics2D` class.

For now, let's display the chart in a frame on the screen. The `ChartFrame` class contains the machinery (a `ChartPanel`) required to display charts:

```
// create and display a frame...
ChartFrame frame = new ChartFrame("Test", chart);
frame.pack();
frame.setVisible(true);
```

And that's all there is to it...

4.2.5 The Complete Program

Here is the complete program, so that you can see which packages you need to import and the order of the code fragments given in the preceding sections:

```
package org.jfree.chart.demo;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.DefaultPieDataset;

public class First {

    /**
     * The starting point for the demo.
     *
     * @param args  ignored.
     */
    public static void main(String[] args) {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();
        data.setValue("Category 1", 43.2);
        data.setValue("Category 2", 27.9);
        data.setValue("Category 3", 79.5);

        // create a chart...
```

```
            JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart",
                                                           data,
                                                           true,    // legend?
                                                           true,    // tooltips?
                                                           false);  // URLs?

            // create and display a frame...
            ChartFrame frame = new ChartFrame("First", chart);
            frame.pack();
            frame.setVisible(true);

        }

    }
```

Hopefully this has convinced you that it is not difficult to create and display charts with JFreeChart. Of course, there is much more to learn...

5 Bar Charts

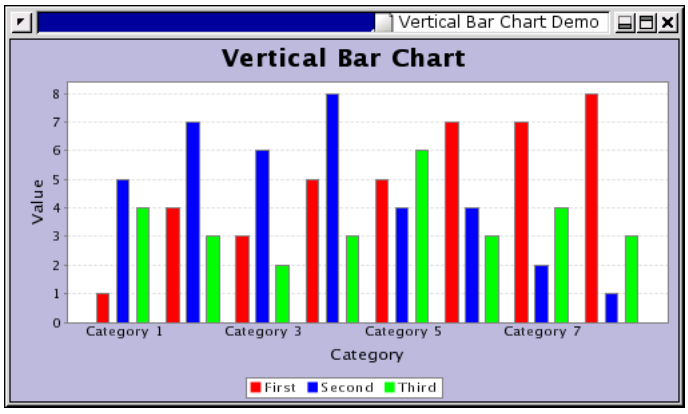
5.1 Introduction

This section describes the *bar charts* that can be created with JFreeChart. Most bar charts are created using data from the `CategoryDataset` interface, but it is also possible to use the `IntervalXYDataset` interface.

5.2 A Vertical Bar Chart

5.2.1 Overview

A *vertical bar chart* is created using data from a `CategoryDataset`, and represents each data item as an "upright" bar. This section presents a sample application that generates the following chart:



The full source code is included in the download (`VerticalBarChartDemo`).

5.2.2 The Dataset

The first step in generating the chart is to create a dataset. In the example, the `DefaultCategoryDataset` class is used:

```
// row keys...
String series1 = "First";
String series2 = "Second";
String series3 = "Third";

// column keys...
String category1 = "Category 1";
String category2 = "Category 2";
String category3 = "Category 3";
String category4 = "Category 4";
String category5 = "Category 5";
String category6 = "Category 6";
String category7 = "Category 7";
String category8 = "Category 8";

// create the dataset...
DefaultCategoryDataset dataset = new DefaultCategoryDataset();

dataset.addValue(1.0, series1, category1);
dataset.addValue(4.0, series1, category2);
dataset.addValue(3.0, series1, category3);
```

```
dataset.addValue(5.0, series1, category4);
dataset.addValue(5.0, series1, category5);
dataset.addValue(7.0, series1, category6);
dataset.addValue(7.0, series1, category7);
dataset.addValue(8.0, series1, category8);

dataset.addValue(5.0, series2, category1);
dataset.addValue(7.0, series2, category2);
dataset.addValue(6.0, series2, category3);
dataset.addValue(8.0, series2, category4);
dataset.addValue(4.0, series2, category5);
dataset.addValue(4.0, series2, category6);
dataset.addValue(2.0, series2, category7);
dataset.addValue(1.0, series2, category8);

dataset.addValue(4.0, series3, category1);
dataset.addValue(3.0, series3, category2);
dataset.addValue(2.0, series3, category3);
dataset.addValue(3.0, series3, category4);
dataset.addValue(6.0, series3, category5);
dataset.addValue(3.0, series3, category6);
dataset.addValue(4.0, series3, category7);
dataset.addValue(3.0, series3, category8);
```

Note that you can use *any* implementation of the `CategoryDataset` interface as your dataset.

5.2.3 Constructing the Chart

The `createVerticalBarChart(...)` method in the `ChartFactory` class provides a convenient way to create the chart:

```
// create the chart...
JFreeChart chart = ChartFactory.createVerticalBarChart(
                                    "Vertical Bar Chart",  // chart title
                                    "Category",            // domain axis label
                                    "Value",               // range axis label
                                    dataset,               // data
                                    true,                  // include legend
                                    true,
                                    false
                                );
```

This method constructs a `JFreeChart` object with a title, legend, and plot with appropriate axes, renderer and tooltip generator. The `dataset` is the one created in the previous section.

5.2.4 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the chart background color;
- the "skip labels" flag is set to true, which means that some category axis labels may be skipped to prevent overlapping;
- the "auto tick units" on the range axis (so that the tick labels always display integer values);

Changing the chart's background color is simple, because this is an attribute maintained by the `JFreeChart` class:

```
// set the background color for the chart...
chart.setBackgroundPaint(new Color(0xBBBBDD));
```

To change other attributes, we first need to obtain a reference to the `CategoryPlot` object used by the chart:

```
CategoryPlot plot = chart.getCategoryPlot();
```

The domain axis (an instance of `HorizontalCategoryAxis`) is modified so that category labels will be skipped (if necessary) to prevent overlapping:

```
// skip some labels if they overlap...
HorizontalCategoryAxis domainAxis = (HorizontalCategoryAxis) plot.getDomainAxis();
domainAxis.setSkipCategoryLabelsToFit(true);
```

Finally, the range axis is modified so that the tick units are always integers:

```
// change the auto tick unit selection to integer units only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(TickUnits.createIntegerTickUnits());
```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to a vertical bar plot.

5.2.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart distribution.

```
package org.jfree.chart.demo;

import java.awt.Color;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.HorizontalCategoryAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class VerticalBarChartDemo extends ApplicationFrame {

    public VerticalBarChartDemo(String title) {

        super(title);

        // row keys...
        String series1 = "First";
        String series2 = "Second";
        String series3 = "Third";

        // column keys...
        String category1 = "Category 1";
        String category2 = "Category 2";
        String category3 = "Category 3";
        String category4 = "Category 4";
        String category5 = "Category 5";
        String category6 = "Category 6";
        String category7 = "Category 7";
        String category8 = "Category 8";
```

```
        // create the dataset...
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        dataset.addValue(1.0, series1, category1);
        dataset.addValue(4.0, series1, category2);
        dataset.addValue(3.0, series1, category3);
        dataset.addValue(5.0, series1, category4);
        dataset.addValue(5.0, series1, category5);
        dataset.addValue(7.0, series1, category6);
        dataset.addValue(7.0, series1, category7);
        dataset.addValue(8.0, series1, category8);

        dataset.addValue(5.0, series2, category1);
        dataset.addValue(7.0, series2, category2);
        dataset.addValue(6.0, series2, category3);
        dataset.addValue(8.0, series2, category4);
        dataset.addValue(4.0, series2, category5);
        dataset.addValue(4.0, series2, category6);
        dataset.addValue(2.0, series2, category7);
        dataset.addValue(1.0, series2, category8);

        dataset.addValue(4.0, series3, category1);
        dataset.addValue(3.0, series3, category2);
        dataset.addValue(2.0, series3, category3);
        dataset.addValue(3.0, series3, category4);
        dataset.addValue(6.0, series3, category5);
        dataset.addValue(3.0, series3, category6);
        dataset.addValue(4.0, series3, category7);
        dataset.addValue(3.0, series3, category8);

        // create the chart...
        JFreeChart chart = ChartFactory.createVerticalBarChart(
                               "Vertical Bar Chart",  // chart title
                               "Category",            // domain axis label
                               "Value",               // range axis label
                               dataset,               // data
                               true,                  // include legend
                               true,
                               false
                           );

        // set the background color for the chart...
        chart.setBackgroundPaint(new Color(0xBBBBDD));

        // get a reference to the plot for further customisation...
        CategoryPlot plot = chart.getCategoryPlot();

        // skip some labels if they overlap...
        HorizontalCategoryAxis domainAxis = (HorizontalCategoryAxis) plot.getDomainAxis();
        domainAxis.setSkipCategoryLabelsToFit(true);

        // set the range axis to display integers only...
        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

        // add the chart to a panel...
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        setContentPane(chartPanel);

    }

    public static void main(String[] args) {

        VerticalBarChartDemo demo = new VerticalBarChartDemo("Vertical Bar Chart Demo");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

5.3 Customising Bar Charts

This section describes some of the methods you can use to customise the appearance of bar charts.

5.3.1 Bar Colors

You can customise the colors used in a bar chart in the same way that you would for most other chart types. You need to obtain a reference to the renderer (the object responsible for drawing the bars in the chart) and set the series colors there:

```
CategoryPlot plot = myChart.getCategoryPlot();
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setSeriesPaint(0, Color.red);
renderer.setSeriesPaint(1, Color.green);
renderer.setSeriesPaint(2, Color.blue);
```

The `setSeriesPaint(...)` method is defined in the `AbstractRenderer` class.

5.3.2 Bar Spacing

JFreeChart allows you to configure the way that bars are distributed along the category axis. There are settings for:

- the margin before the start of the first category;
- the margin between categories;
- the margin after the end of the last category;
- the gap between bars within a category;

The first three items are configured using the `CategoryAxis`:

```
CategoryPlot plot = myChart.getCategoryPlot();
CategoryAxis axis = plot.getDomainAxis();
axis.setLowerMargin(0.02);  // two percent
axis.setCategoryMargin(0.10);  // ten percent
axis.setUpperMargin(0.02);  // two percent
```

All of the margins are specified as a percentage of the length of the category axis, to allow for the fact that JFreeChart can draw charts at varying sizes. Note that the percentage for the category margin specifies the total margin for all the categories—if *N* is the number of categories, the margin is allocated over *N - 1* gaps between the categories.

The spacing between bars *within a category* is not controlled by the axis—instead, it is dealt with by the `BarRenderer`.

```
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setItemMargin(0.15);  // fifteen percent
```

As with the category margin, the item margin is the total margin for all the "intra-category" gaps in the chart. If there are *M* series in the chart, and *N* categories, then there will be *(M - 1)(N - 1)* gaps.

A final point to note—the bar widths are dynamically calculated to fill the remaining space after the various margins have been allocated. If is not possible to specify fixed bar widths in JFreeChart.

6 Line Charts

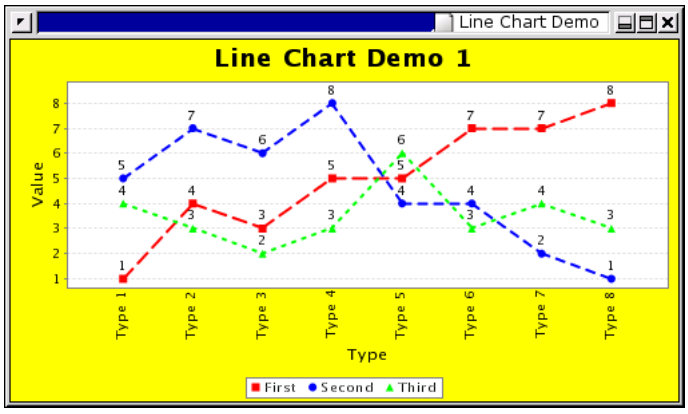
6.1 Introduction

This section describes the *line charts* that can be created with JFreeChart. It is possible to create line charts using data from either the `CategoryDataset` interface or the `XYDataset` interface.

6.2 A Line Chart Based On A Category Dataset

6.2.1 Overview

A *line chart* based on a `CategoryDataset` simply connects each *(category, value)* data item using straight lines. This section presents a sample application that generates the following chart:



The full source code is included in the download (`LineChartDemo1`).

6.2.2 The Dataset

The first step in generating the chart is, as always, to create a dataset. In the example, the `DefaultCategoryDataset` class is used:

```
// row keys...
String series1 = "First";
String series2 = "Second";
String series3 = "Third";

// column keys...
String type1 = "Type 1";
String type2 = "Type 2";
String type3 = "Type 3";
String type4 = "Type 4";
String type5 = "Type 5";
String type6 = "Type 6";
String type7 = "Type 7";
String type8 = "Type 8";

// create the dataset...
DefaultCategoryDataset dataset = new DefaultCategoryDataset();

dataset.addValue(1.0, series1, type1);
dataset.addValue(4.0, series1, type2);
dataset.addValue(3.0, series1, type3);
```

```
dataset.addValue(5.0, series1, type4);
dataset.addValue(5.0, series1, type5);
dataset.addValue(7.0, series1, type6);
dataset.addValue(7.0, series1, type7);
dataset.addValue(8.0, series1, type8);

dataset.addValue(5.0, series2, type1);
dataset.addValue(7.0, series2, type2);
dataset.addValue(6.0, series2, type3);
dataset.addValue(8.0, series2, type4);
dataset.addValue(4.0, series2, type5);
dataset.addValue(4.0, series2, type6);
dataset.addValue(2.0, series2, type7);
dataset.addValue(1.0, series2, type8);

dataset.addValue(4.0, series3, type1);
dataset.addValue(3.0, series3, type2);
dataset.addValue(2.0, series3, type3);
dataset.addValue(3.0, series3, type4);
dataset.addValue(6.0, series3, type5);
dataset.addValue(3.0, series3, type6);
dataset.addValue(4.0, series3, type7);
dataset.addValue(3.0, series3, type8);
```

Note that you can use *any* implementation of the `CategoryDataset` interface as your dataset.

6.2.3 Constructing the Chart

The `createLineChart(...)` method in the `ChartFactory` class provides a convenient way to create the chart:

```
JFreeChart chart = ChartFactory.createLineChart("Line Chart Demo 1",
                                    "Type",               // domain axis label
                                    "Value",              // range axis label
                                    dataset,              // data
                                    true,                 // include legend
                                    true,                 // tooltips
                                    false                 // urls
                                );
```

This method constructs a `JFreeChart` object with a title, legend, and plot with appropriate axes, renderer and tooltip generator. The `dataset` is the one created in the previous section.

6.2.4 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the chart background color;
- the series stroke;
- the "auto tick units" on the range axis (so that the tick labels always display integer values);

Changing the chart's background color is simple, because this is an attribute maintained by the `JFreeChart` class:

```
// set the background color for the chart...
chart.setBackgroundPaint(Color.yellow);
```

To change other attributes, we first need to obtain a reference to the `CategoryPlot` object used by the chart:

```
CategoryPlot plot = chart.getCategoryPlot();
```

The plot is responsible for drawing the data and axes on the chart. Some of this work is delegated to a *renderer*, which you can access via the `getRenderer()` method. The renderer maintains most of the attributes that relate to the appearance of the data items within the chart. To modify the line stroke used for each series:

```
// set the stroke for each series...
plot.getRenderer().setSeriesStroke(0,
    new BasicStroke(2.0f,
                    BasicStroke.CAP_ROUND,
                    BasicStroke.JOIN_ROUND,
                    1.0f,
                    new float[] { 10.0f, 6.0f },
                    0.0f));
plot.getRenderer().setSeriesStroke(1,
    new BasicStroke(2.0f,
                    BasicStroke.CAP_ROUND,
                    BasicStroke.JOIN_ROUND,
                    1.0f,
                    new float[] { 6.0f, 6.0f },
                    0.0f));
plot.getRenderer().setSeriesStroke(2,
    new BasicStroke(2.0f,
                    BasicStroke.CAP_ROUND,
                    BasicStroke.JOIN_ROUND,
                    1.0f,
                    new float[] { 2.0f, 6.0f },
                    0.0f));
```

The plot also manages the chart's axes. In the example, the range axis is modified so that it only displays integer values for the tick labels:

```
// change the auto tick unit selection to integer units only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(TickUnits.createIntegerTickUnits());
```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to a line plot.

6.2.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart download.

```
package org.jfree.chart.demo;

import java.awt.BasicStroke;
import java.awt.Color;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.StandardLegend;
import org.jfree.chart.axis.HorizontalCategoryAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.renderer.LineAndShapeRenderer;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
```

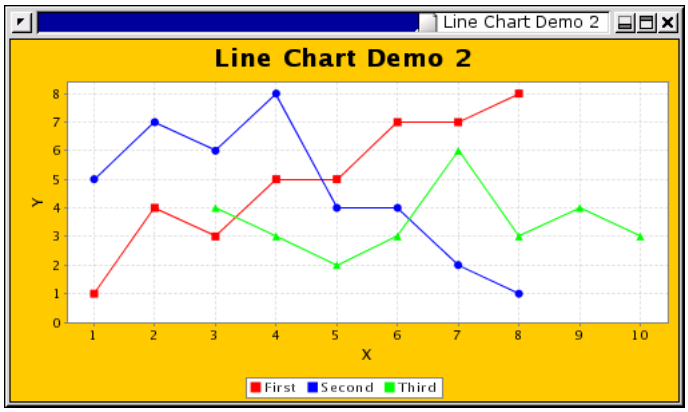
```
import org.jfree.ui.RefineryUtilities;

public class LineChartDemo1 extends ApplicationFrame {

    public LineChartDemo1(String title) {

        super(title);

        // row keys...
        String series1 = "First";
        String series2 = "Second";
        String series3 = "Third";

        // column keys...
        String type1 = "Type 1";
        String type2 = "Type 2";
        String type3 = "Type 3";
        String type4 = "Type 4";
        String type5 = "Type 5";
        String type6 = "Type 6";
        String type7 = "Type 7";
        String type8 = "Type 8";

        // create the dataset...
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        dataset.addValue(1.0, series1, type1);
        dataset.addValue(4.0, series1, type2);
        dataset.addValue(3.0, series1, type3);
        dataset.addValue(5.0, series1, type4);
        dataset.addValue(5.0, series1, type5);
        dataset.addValue(7.0, series1, type6);
        dataset.addValue(7.0, series1, type7);
        dataset.addValue(8.0, series1, type8);

        dataset.addValue(5.0, series2, type1);
        dataset.addValue(7.0, series2, type2);
        dataset.addValue(6.0, series2, type3);
        dataset.addValue(8.0, series2, type4);
        dataset.addValue(4.0, series2, type5);
        dataset.addValue(4.0, series2, type6);
        dataset.addValue(2.0, series2, type7);
        dataset.addValue(1.0, series2, type8);

        dataset.addValue(4.0, series3, type1);
        dataset.addValue(3.0, series3, type2);
        dataset.addValue(2.0, series3, type3);
        dataset.addValue(3.0, series3, type4);
        dataset.addValue(6.0, series3, type5);
        dataset.addValue(3.0, series3, type6);
        dataset.addValue(4.0, series3, type7);
        dataset.addValue(3.0, series3, type8);

        // create the chart...
        JFreeChart chart = ChartFactory.createLineChart(
            "Line Chart Demo 1",  // chart title
            "Type",               // domain axis label
            "Value",              // range axis label
            dataset,              // data
            true,                 // include legend
            true,                 // tooltips
            false                 // urls
        );

        // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
        StandardLegend legend = (StandardLegend) chart.getLegend();
        legend.setDisplaySeriesShapes(true);

        chart.setBackgroundPaint(Color.yellow);

        CategoryPlot plot = chart.getCategoryPlot();

        // set the stroke for each series...
        plot.getRenderer().setSeriesStroke(
```

```
            0, new BasicStroke(2.0f,
                               BasicStroke.CAP_ROUND,
                               BasicStroke.JOIN_ROUND,
                               1.0f,
                               new float[] { 10.0f, 6.0f },
                               0.0f)
        );
        plot.getRenderer().setSeriesStroke(
            1, new BasicStroke(2.0f,
                               BasicStroke.CAP_ROUND,
                               BasicStroke.JOIN_ROUND,
                               1.0f,
                               new float[] { 6.0f, 6.0f },
                               0.0f)
        );
        plot.getRenderer().setSeriesStroke(
            2, new BasicStroke(2.0f,
                               BasicStroke.CAP_ROUND,
                               BasicStroke.JOIN_ROUND,
                               1.0f,
                               new float[] { 2.0f, 6.0f },
                               0.0f)
        );

        // label data points with values...
        plot.setValueLabelsVisible(true);

        // add a range marker...
        //plot.addRangeMarker(new Marker(8.0));

        // customise the renderer...
        LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.getRenderer();
        renderer.setDrawShapes(true);

        // customise the domain axis...
        HorizontalCategoryAxis domainAxis = (HorizontalCategoryAxis) plot.getDomainAxis();
        domainAxis.setVerticalCategoryLabels(true);

        // customise the range axis...
        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
        rangeAxis.setAutoRangeIncludesZero(false);
        rangeAxis.setUpperMargin(0.12);

        // OPTIONAL CUSTOMISATION COMPLETED.

        // add the chart to a panel...
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        setContentPane(chartPanel);

    }

    public static void main(String[] args) {

        LineChartDemo1 demo = new LineChartDemo1("Line Chart Demo");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

6.3 A Line Chart Based On An XYDataset

6.3.1 Overview

A *line chart* based on an `XYDataset` connects each *(x, y)* point with a straight line. This section presents a sample application that generates the following chart:



The complete source code (`LineChartDemo2`) is distributed with JFreeChart.

6.3.2 The Dataset

For this chart, an `XYSeriesCollection` is used as the dataset (you can use any implementation of the `XYDataset` interface):

```
// create a dataset...
XYSeries series1 = new XYSeries("First");
series1.add(1.0, 1.0);
series1.add(2.0, 4.0);
series1.add(3.0, 3.0);
series1.add(4.0, 5.0);
series1.add(5.0, 5.0);
series1.add(6.0, 7.0);
series1.add(7.0, 7.0);
series1.add(8.0, 8.0);

XYSeries series2 = new XYSeries("Second");
series2.add(1.0, 5.0);
series2.add(2.0, 7.0);
series2.add(3.0, 6.0);
series2.add(4.0, 8.0);
series2.add(5.0, 4.0);
series2.add(6.0, 4.0);
series2.add(7.0, 2.0);
series2.add(8.0, 1.0);

XYSeries series3 = new XYSeries("Third");
series3.add(3.0, 4.0);
series3.add(4.0, 3.0);
series3.add(5.0, 2.0);
series3.add(6.0, 3.0);
series3.add(7.0, 6.0);
series3.add(8.0, 3.0);
series3.add(9.0, 4.0);
series3.add(10.0, 3.0);

XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(series1);
dataset.addSeries(series2);
dataset.addSeries(series3);
```

Notice how each series has x-values (not just y-values) that are independent from the other series.

6.3.3 Constructing the Chart

The `createLineXYChart(...)` method in the `ChartFactory` class provides a convenient way to create the chart:

```
// create the chart...
JFreeChart chart = ChartFactory.createLineXYChart(
                                    "Line Chart Demo 2",  // chart title
                                    "X",                  // x axis label
                                    "Y",                  // y axis label
                                    dataset,              // data
                                    true,                 // include legend
                                    true,                 // tooltips
                                    false                 // urls
                                );
```

This method constructs a `JFreeChart` object with a title, legend and plot with appropriate axes and renderer. The `dataset` is the one created in the previous section.

6.3.4 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the chart background color;
- the renderer is modified to draw shapes as well as lines;
- the tick unit collection for the vertical axis, so that the tick values always display integer values;

Changing the chart's background color is simple:

```
// set the background color for the chart...
chart.setBackgroundPaint(Color.orange);
```

The renderer is modified to display filled shapes in addition to the default lines:

```
// get a reference to the plot for further customisation...
XYPlot plot = chart.getXYPlot();
StandardXYItemRenderer renderer = (StandardXYItemRenderer) plot.getRenderer();
renderer.setPlotShapes(true);
renderer.setDefaultShapeFilled(true);
```

The final modification is a change to the range axis. We change the default collection of tick units (which allow fractional values) to an integer-only collection:

```
// change the auto tick unit selection to integer units only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to an `XYPlot`.

6.3.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart download.

```
package org.jfree.chart.demo;

import java.awt.Color;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.StandardXYItemRenderer;
import org.jfree.data.XYSeries;
import org.jfree.data.XYSeriesCollection;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class LineChartDemo2 extends ApplicationFrame {

    /**
     * Creates a new demo.
     *
     * @param title  the frame title.
     */
    public LineChartDemo2(String title) {

        super(title);

        // create a dataset...
        XYSeries series1 = new XYSeries("First");
        series1.add(1.0, 1.0);
        series1.add(2.0, 4.0);
        series1.add(3.0, 3.0);
        series1.add(4.0, 5.0);
        series1.add(5.0, 5.0);
        series1.add(6.0, 7.0);
        series1.add(7.0, 7.0);
        series1.add(8.0, 8.0);

        XYSeries series2 = new XYSeries("Second");
        series2.add(1.0, 5.0);
        series2.add(2.0, 7.0);
        series2.add(3.0, 6.0);
        series2.add(4.0, 8.0);
        series2.add(5.0, 4.0);
        series2.add(6.0, 4.0);
        series2.add(7.0, 2.0);
        series2.add(8.0, 1.0);

        XYSeries series3 = new XYSeries("Third");
        series3.add(3.0, 4.0);
        series3.add(4.0, 3.0);
        series3.add(5.0, 2.0);
        series3.add(6.0, 3.0);
        series3.add(7.0, 6.0);
        series3.add(8.0, 3.0);
        series3.add(9.0, 4.0);
        series3.add(10.0, 3.0);

        XYSeriesCollection dataset = new XYSeriesCollection();
        dataset.addSeries(series1);
        dataset.addSeries(series2);
        dataset.addSeries(series3);

        // create the chart...
        JFreeChart chart = ChartFactory.createLineXYChart(
            "Line Chart Demo 2",  // chart title
            "X",                  // x axis label
            "Y",                  // y axis label
            dataset,              // data
            true,                 // include legend
            true,                 // tooltips
            false                 // urls
        );

        // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
```

```
        chart.setBackgroundPaint(Color.orange);

        // get a reference to the plot for further customisation...
        XYPlot plot = chart.getXYPlot();
        StandardXYItemRenderer renderer = (StandardXYItemRenderer) plot.getRenderer();
        renderer.setPlotShapes(true);
        renderer.setDefaultShapeFilled(true);

        // change the auto tick unit selection to integer units only...
        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

        // OPTIONAL CUSTOMISATION COMPLETED.

        // add the chart to a panel...
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        setContentPane(chartPanel);

    }

    public static void main(String[] args) {

        LineChartDemo2 demo = new LineChartDemo2("Line Chart Demo 2");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

7 Time Series Charts

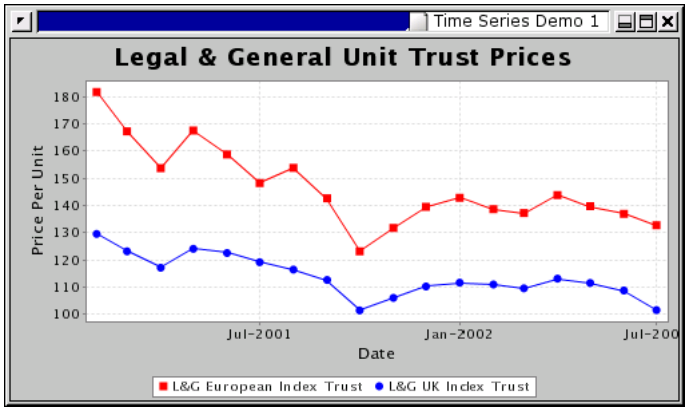
7.1 Introduction

Time series charts are very similar to line charts, except that the values on the domain axis are dates rather than numbers. This section describes how to create time series charts with JFreeChart.

7.2 Time Series Charts

7.2.1 Overview

A *time series chart* is really just a *line chart* using data obtained via the `XYDataset` interface (see the example in the previous section). The difference is that the x-values are displayed as dates on the domain axis. This section presents a sample application that generates the following chart:



The complete source code (`TimeSeriesDemo`) for this example is included in the JFreeChart distribution.

7.2.2 Dates or Numbers?

Time series charts are created using data from an `XYDataset`. This interface doesn't have any methods that return dates, so how does JFreeChart create time series charts?

The x-values returned by the dataset are `Number` objects, but the values are interpreted in a special way—they are assumed to represent the number of milliseconds since midnight, 1 January 1970 (the encoding used by the `java.util.Date` class).

A special axis class (`DateAxis`) converts from milliseconds to dates and back again as necessary, allowing the axis to display tick labels formatted as dates.

7.2.3 The Dataset

For the demo chart, a `TimeSeriesCollection` is used as the dataset (you can use any implementation of the `XYDataset` interface):

```
TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
s1.add(new Month(2, 2001), 181.8);
s1.add(new Month(3, 2001), 167.3);
s1.add(new Month(4, 2001), 153.8);
s1.add(new Month(5, 2001), 167.6);
s1.add(new Month(6, 2001), 158.8);
s1.add(new Month(7, 2001), 148.3);
s1.add(new Month(8, 2001), 153.9);
s1.add(new Month(9, 2001), 142.7);
s1.add(new Month(10, 2001), 123.2);
s1.add(new Month(11, 2001), 131.8);
s1.add(new Month(12, 2001), 139.6);
s1.add(new Month(1, 2002), 142.9);
s1.add(new Month(2, 2002), 138.7);
s1.add(new Month(3, 2002), 137.3);
s1.add(new Month(4, 2002), 143.9);
s1.add(new Month(5, 2002), 139.8);
s1.add(new Month(6, 2002), 137.0);
s1.add(new Month(7, 2002), 132.8);

TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
s2.add(new Month(2, 2001), 129.6);
s2.add(new Month(3, 2001), 123.2);
s2.add(new Month(4, 2001), 117.2);
s2.add(new Month(5, 2001), 124.1);
s2.add(new Month(6, 2001), 122.6);
s2.add(new Month(7, 2001), 119.2);
s2.add(new Month(8, 2001), 116.5);
s2.add(new Month(9, 2001), 112.7);
s2.add(new Month(10, 2001), 101.5);
s2.add(new Month(11, 2001), 106.1);
s2.add(new Month(12, 2001), 110.3);
s2.add(new Month(1, 2002), 111.7);
s2.add(new Month(2, 2002), 111.0);
s2.add(new Month(3, 2002), 109.6);
s2.add(new Month(4, 2002), 113.2);
s2.add(new Month(5, 2002), 111.6);
s2.add(new Month(6, 2002), 108.8);
s2.add(new Month(7, 2002), 101.6);

TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(s1);
dataset.addSeries(s2);
```

In the example, the series contain monthly data. However, the `TimeSeries` class can be used to represent values observed at other intervals (annual, daily, hourly etc).

7.2.4 Constructing the Chart

The `createTimeSeriesChart(...)` method in the `ChartFactory` class provides a convenient way to create the chart:

```
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    chartTitle,
    "Date", "Price Per Unit",
    dataset,
    true,
    true,
    false
);
```

This method constructs a `JFreeChart` object with a title, legend and plot with appropriate axes and renderer. The `dataset` is the one created in the previous section.

7.2.5 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the renderer is changed to display series shapes at each data point, in addition to the lines between data points;
- the legend is set up to display the series shapes;
- a date format override is set for the domain axis;

Modifying the renderer requires a couple of steps to obtain a reference to the renderer and then cast it to a `StandardXYItemRenderer`:

```
XYPlot plot = chart.getXYPlot();
XYItemRenderer renderer = plot.getRenderer();
if (renderer instanceof StandardXYItemRenderer) {
    StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
    rr.setPlotShapes(true);
    rr.setDefaultShapeFilled(true);
}
```

Similarly, the legend must be cast to a `StandardLegend`, before setting the flag that tells the legend to display shapes as the series keys:

```
StandardLegend sl = (StandardLegend) chart.getLegend();
sl.setDisplaySeriesShapes(true);
```

In the final customisation, a date format override is set for the domain axis.

```
DateAxis axis = (DateAxis) plot.getDomainAxis();
axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));
```

When this is set, the axis will continue to "auto-select" a `DateTickUnit` from the collection of standard tick units, but it will ignore the formatting from the tick unit and use the override format instead.

7.2.6 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart download.

```
package org.jfree.chart.demo;

import java.text.SimpleDateFormat;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.StandardLegend;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.StandardXYItemRenderer;
import org.jfree.chart.renderer.XYItemRenderer;
import org.jfree.data.XYDataset;
import org.jfree.data.time.Month;
import org.jfree.data.time.TimeSeries;
```

```
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

public class TimeSeriesDemo extends ApplicationFrame {

    public TimeSeriesDemo(String title) {

        super(title);

        // create a title...
        String chartTitle = "Legal & General Unit Trust Prices";
        XYDataset dataset = createDataset();

        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            chartTitle,
            "Date", "Price Per Unit",
            dataset,
            true,
            true,
            false
        );

        StandardLegend sl = (StandardLegend) chart.getLegend();
        sl.setDisplaySeriesShapes(true);

        XYPlot plot = chart.getXYPlot();
        XYItemRenderer renderer = plot.getRenderer();
        if (renderer instanceof StandardXYItemRenderer) {
            StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
            rr.setPlotShapes(true);
            rr.setDefaultShapeFilled(true);
        }
        DateAxis axis = (DateAxis) plot.getDomainAxis();
        axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));

        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        chartPanel.setMouseZoomable(true, false);
        setContentPane(chartPanel);

    }

    public XYDataset createDataset() {

        TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
        s1.add(new Month(2, 2001), 181.8);
        s1.add(new Month(3, 2001), 167.3);
        s1.add(new Month(4, 2001), 153.8);
        s1.add(new Month(5, 2001), 167.6);
        s1.add(new Month(6, 2001), 158.8);
        s1.add(new Month(7, 2001), 148.3);
        s1.add(new Month(8, 2001), 153.9);
        s1.add(new Month(9, 2001), 142.7);
        s1.add(new Month(10, 2001), 123.2);
        s1.add(new Month(11, 2001), 131.8);
        s1.add(new Month(12, 2001), 139.6);
        s1.add(new Month(1, 2002), 142.9);
        s1.add(new Month(2, 2002), 138.7);
        s1.add(new Month(3, 2002), 137.3);
        s1.add(new Month(4, 2002), 143.9);
        s1.add(new Month(5, 2002), 139.8);
```

```
        s1.add(new Month(6, 2002), 137.0);
        s1.add(new Month(7, 2002), 132.8);

        TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
        s2.add(new Month(2, 2001), 129.6);
        s2.add(new Month(3, 2001), 123.2);
        s2.add(new Month(4, 2001), 117.2);
        s2.add(new Month(5, 2001), 124.1);
        s2.add(new Month(6, 2001), 122.6);
        s2.add(new Month(7, 2001), 119.2);
        s2.add(new Month(8, 2001), 116.5);
        s2.add(new Month(9, 2001), 112.7);
        s2.add(new Month(10, 2001), 101.5);
        s2.add(new Month(11, 2001), 106.1);
        s2.add(new Month(12, 2001), 110.3);
        s2.add(new Month(1, 2002), 111.7);
        s2.add(new Month(2, 2002), 111.0);
        s2.add(new Month(3, 2002), 109.6);
        s2.add(new Month(4, 2002), 113.2);
        s2.add(new Month(5, 2002), 111.6);
        s2.add(new Month(6, 2002), 108.8);
        s2.add(new Month(7, 2002), 101.6);

        TimeSeriesCollection dataset = new TimeSeriesCollection();
        dataset.addSeries(s1);
        dataset.addSeries(s2);

        return dataset;

    }

    public static void main(String[] args) {

        TimeSeriesDemo demo = new TimeSeriesDemo("Time Series Demo 1");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

8 Customising Charts

8.1 Introduction

JFreeChart has been designed to be highly customisable. There are many attributes that you can set to change the default appearance of your charts. In this section, some common techniques for customising charts are presented.

8.2 Chart Attributes

8.2.1 Overview

At the highest level, you can customise the appearance of your charts using methods in the `JFreeChart` class. This allows you to control:

- the chart title and sub-titles;
- the background color and/or image;
- whether or not *anti-aliasing* is used when drawing charts;

These items are described in the following sections.

8.2.2 The Chart Title

A chart has one title that can appear at the top, bottom, left or right of the chart. The title is an instance of `TextTitle`. You can obtain a reference to the title using the `getTitle()` method:

```
TextTitle title = myChart.getTitle();
```

To modify the title text (without changing the font or position):

```
myChart.setTitle("A Chart Title");
```

The placement of the title at the top, bottom, left or right of the chart is controlled by a property of the title itself. To move the title to the bottom of the chart:

```
chart.getTitle().setPosition(AbstractTitle.BOTTOM);
```

If you don't want a title to appear on your chart, set it to `null`.

8.2.3 Subtitles

A chart can have any number of subtitles. To add a sub-title to a chart, create a subtitle (any subclass of `AbstractTitle`) and add it to the chart. For example:

```
TextTitle subtitle1 = new TextTitle("A Subtitle");
myChart.addSubtitle();
```

You can add as many sub-titles as you like to a chart, but keep in mind that as you add more sub-titles there will be less and less space available for drawing the chart.

To modify an existing sub-title, you need to get a reference to the sub-title. For example:

```
AbstractTitle subtitle = myChart.getSubtitle(0);
```

You will need to cast the `AbstractTitle` reference to an appropriate subclass before you can change its properties.

You can check the number of sub-titles using the `getSubtitleCount()` method.

8.2.4 Setting the Background Color

You can use the `setBackgroundPaint(...)` method to set the background color for a chart.[4] For example:

```
myChart.setBackgroundPaint(Color.blue);
```

You can use any implementation of the `Paint` interface, including the Java classes `Color`, `GradientPaint` and `TexturePaint`. For example:

```
Paint p = new GradientPaint(0, 0, Color.white, 1000, 0, Color.green));
myChart.setBackgroundPaint(p);
```

You can also set the background paint to `null`, which is recommended if you have specified a background image for your chart.

8.2.5 Using a Background Image

You can use the `setBackgroundImage(...)` method to set a background image for a chart.

```
myChart.setBackgroundImage(JFreeChart.INFO.getLogo());
```

By default, the image will be scaled to fit the area that the chart is being drawn into, but you can change this using the `setBackgroundImageAlignment(...)` method.

```
myChart.setBackgroundImageAlignment(Align.TOP_LEFT);
```

Using the `setBackgroundImageAlpha(...)` method, you can control the alpha-transparency for the image.

If you want an image to fill only the *data area* of your chart (that is, the area inside the axes), then you need to add a background image to the chart's `Plot` (described later).

8.2.6 Antialiasing

JFreeChart makes use of the Java2D antialiasing feature to draw smooth looking charts. You can switch this feature on or off using the `setAntiAlias(boolean)` method:

```
// turn on antialiasing...
chart.setAntiAlias(true);
```

By default, charts are drawn with anti-aliasing turned on.

[4] You can also set the background color for the chart's plot area, which has a slightly different effect—refer to the `Plot` class for details.

8.3 Plot Attributes

8.3.1 Overview

The `JFreeChart` class delegates a lot of the work in drawing a chart to the `Plot` class (or, rather, to a specific subclass of `Plot`). The `getPlot()` method in the `JFreeChart` class returns a reference to the plot being used by the chart.

```
Plot plot = myChart.getPlot();
```

You may need to cast this reference to a specific subclass of `Plot`, for example:

```
CategoryPlot plot = myChart.getCategoryPlot();
```

...or:

```
XYPlot plot = myChart.getXYPlot();
```

Note that these methods will throw a `ClassCastException` if the plot is not an appropriate class.

8.3.2 Which Plot Subclass?

How do you know which subclass of `Plot` is being used by a chart? As you gain experience with JFreeChart, it will become clear which charts use `CategoryPlot` and which charts use `XYPlot`. If in doubt, take a look in the `ChartFactory` class source code to see how each chart type is put together.

8.3.3 Setting the Background Paint

You can use the `setBackgroundPaint(...)` method to set the background color for a plot. For example:

```
Plot plot = myChart.getPlot();
plot.setBackgroundPaint(Color.white);
```

You can use any implementation of the `Paint` interface, including the Java classes `Color`, `GradientPaint` and `TexturePaint`. You can also set the background paint to `null`.

8.3.4 Using a Background Image

You can use the `setBackgroundImage(...)` method to set a background image for a plot:

```
Plot plot = myChart.getPlot();
plot.setBackgroundImage(JFreeChart.INFO.getLogo());
```

By default, the image will be scaled to fit the area that the plot is being drawn into. You can change this using the `setBackgroundImageAlignment(...)` method:

```
plot.setBackgroundImageAlignment(Align.BOTTOM_RIGHT);
```

Use the `setBackgroundAlpha(...)` method to control the alpha-transparency used for the image.

If you prefer your image to fill the entire chart area, then you need to add a background image to the `JFreeChart` object (described previously).

8.4 Axis Attributes

8.4.1 Overview

The majority of charts created with JFreeChart have two axes, a *domain axis* and a *range axis*. Of course, there are some charts (for example, pie charts) that don't have axes at all. For charts where axes are used, the `Axis` objects are managed by the `Plot`.

8.4.2 Obtaining an Axis Reference

Before you can change the properties of an axis, you need to obtain a reference to the axis. The plot classes `CategoryPlot` and `XYPlot` both have methods `getDomainAxis()` and `getRangeAxis()`.

These methods return a reference to a `ValueAxis`, except in the case of the `CategoryPlot`, where the *domain axis* is an instance of `CategoryAxis`.

```
// get an axis reference...
CategoryPlot myPlot = myChart.getCategoryPlot();
CategoryAxis domainAxis = myPlot.getDomainAxis();

// change axis properties...
domainAxis.setLabel("Categories");
domainAxis.setLabelFont(someFont);
```

There are many different subclasses of the `CategoryAxis` and `ValueAxis` classes. Sometimes you will need to cast your axis reference to a more specific subclass, in order to access some of its attributes. For example, if you know that your range axis is a `NumberAxis` (and the range axis almost always is), then you can do the following:

```
XYPlot myPlot = myChart.getXYPlot();
NumberAxis rangeAxis = (NumberAxis) myPlot.getRangeAxis();
rangeAxis.setAutoRange(false);
```

8.4.3 Setting the Axis Label

You can use the `setLabel(...)` method to change the axis label. If you would prefer not to have a label for your axis, just set it to `null`.

You can change the font, color and insets (the space around the outside of the label) with the methods `setLabelFont(...)`, `setLabelPaint(...)`, and `setLabelInsets(...)`, defined in the `Axis` class.

8.4.4 Rotating Axis Labels

For vertical axes (`VerticalCategoryAxis` and `VerticalNumberAxis`), the axis label can be drawn with a vertical orientation to save space:

```
XYPlot plot = myChart.getXYPlot();
VerticalNumberAxis axis = (VerticalNumberAxis) plot.getRangeAxis();
axis.setVerticalLabel(true);
```

The default setting for this flag is `true`.

8.4.5 Rotating Category Labels

The category labels on a `HorizontalCategoryAxis` can be displayed with a vertical orientation, which is useful when the labels overlap because of a lack of space. Use the `setVerticalCategoryLabels(boolean)` method as follows:

```
CategoryPlot plot = myChart.getCategoryPlot();
HorizontalCategoryAxis axis = (HorizontalCategoryAxis) plot.getDomainAxis();
axis.setVerticalCategoryLabels(true);
```

The `HorizontalNumberAxis` and `HorizontalDateAxis` classes have the same feature available via the `setVerticalTickLabels(boolean)` method.

8.4.6 Hiding Tick Labels

To hide the tick labels for an axis:

```
CategoryPlot plot = myChart.getCategoryPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setTickLabelsVisible(false);
```

For a category axis, `setTickLabelsVisible(false)` will hide the category labels.

8.4.7 Hiding Tick Marks

To hide the tick marks for an axis:

```
XYPlot plot = myChart.getXYPlot();
Axis axis = plot.getDomainAxis();
axis.setTickMarksVisible(false);
```

Category axes do not have tick marks.

8.4.8 Setting the Tick Size

By default, numerical and date axes automatically select a tick size so that the tick labels will not overlap. You can override this by setting your own tick unit using the `setTickUnit(...)` method.

Alternatively, for a `NumberAxis` or a `DateAxis` you can specify your own set of tick units from which the axis will automatically select an appropriate tick size. This is described in the following sections.

8.4.9 Specifying "Standard" Number Tick Units

In the `NumberAxis` class, there is a method `setStandardTickUnits(...)` that allows you to supply your own set of tick units for the "auto tick unit selection" mechanism.

One common application is where you have a number axis that should only display integers. In this case, you don't want tick units of (say) 0.5 or 0.25. There is a (static) method in the `NumberAxis` class that returns a set of standard integer tick units:

```
XYPlot myPlot = myChart.getXYPlot();
NumberAxis axis = (NumberAxis) myPlot.getRangeAxis();
TickUnits units = NumberAxis.createIntegerTickUnits();
axis.setStandardTickUnits(units);
```

You are free to create your own `TickUnits` collection, if you want greater control over the standard tick units.

8.4.10 Specifying "Standard" Date Tick Units

Similar to the case in the previous section, the `DateAxis` class has a method `setStandardTickUnits(...)` that allows you to supply your own set of tick units for the "auto tick unit selection" mechanism.

The `createStandardDateTickUnits()` method returns the default collection for a `DateAxis`, but you are free to create your own `TickUnits` collection if you want greater control over the standard tick units.

9 Combined Charts

9.1 Introduction

JFreeChart supports *combined charts* via several plot classes that can manage any number of *sub-plots*:

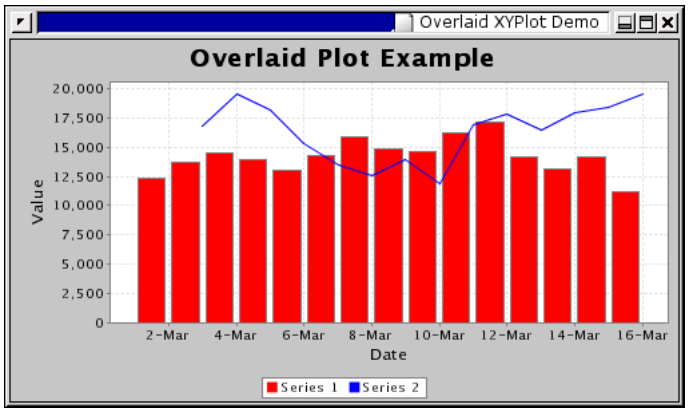
- `CombinedXYPlot`;
- `OverlaidXYPlot`;
- `OverlaidVerticalCategoryPlot`;

In this section, I describe a few examples that use the combined charts facility. These examples are included in the JFreeChart distribution.

9.2 Creating an Overlaid XY Plot

9.2.1 Overview

An *overlaid XY plot* is a special type of plot that combines two or more *sub-plots* (`XYPlot` instances) together on one chart, using shared axes. This section presents a sample application that generates a *vertical XY bar chart* combined with a *time series*:



The complete source code (`OverlaidXYPlotDemo`) is included in the JFreeChart distribution.

9.2.2 The Application

The procedure for creating a chart containing an overlaid plot is not very different from the procedure for creating a standard chart. However, you cannot use the `ChartFactory` class, so you need to be familiar with creating instances of `XYPlot` and `JFreeChart` by calling the constructors directly.

The `TimeSeriesCollection` class does a lot of the background work in the example. It implements both the `XYDataset` interface that is required to create the time series plot, and the `IntervalXYDataset` interface that is required to create the vertical XY bar chart.

In your own code, you may provide your own implementations of these dataset interfaces, but you are also free to use the `TimeSeriesCollection` class if it is convenient for you.

The code for creating the datasets follows a pattern that is used quite frequently in the JFreeChart demonstration code:

```
// create dataset 1...
TimeSeries series1 = new TimeSeries("Series 1", Day.class);
series1.add(new Day(1, SerialDate.MARCH, 2002), 12353.3);
series1.add(new Day(2, SerialDate.MARCH, 2002), 13734.4);
...
series1.add(new Day(15, SerialDate.MARCH, 2002), 11235.2);

return new TimeSeriesCollection(series1);
```

In the demonstration application, one time series collection is assigned to `data1` and another is assigned to `data2`.

9.2.3 Constructing the Chart

With the two datasets `data1` and `data2`, we can proceed to construct the overlaid chart. The first step is to create the two subplots (both with `null` axes):

```
DrawingSupplier supplier = new DefaultDrawingSupplier();

// create subplot 1...
IntervalXYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new VerticalXYBarRenderer(0.20);
renderer1.setDrawingSupplier(supplier);
renderer1.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy", "0.00"));
XYPlot subplot1 = new XYPlot(data1, null, null, renderer1);

// create subplot 2...
XYDataset data2 = createDataset2();
XYItemRenderer renderer2 = new StandardXYItemRenderer();
renderer2.setDrawingSupplier(supplier);
renderer2.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy", "0.00"));
XYPlot subplot2 = new XYPlot(data2, null, null, renderer2);
```

Notice the use of a `DrawingSupplier` to coordinate the colors used by the renderers in the two subplots.

The next step is to create a new *parent plot* (`OverlaidXYPlot`) and add the subplots:

```
// make an overlaid plot and add the subplots...
ValueAxis domainAxis = new HorizontalDateAxis("Date");
ValueAxis rangeAxis = new VerticalNumberAxis("Value");
OverlaidXYPlot plot = new OverlaidXYPlot(domainAxis, rangeAxis);
plot.add(subplot1);
plot.add(subplot2);
```

Note that it is the parent plot that maintains the domain and range axes.

Finally, the `JFreeChart` object is created:

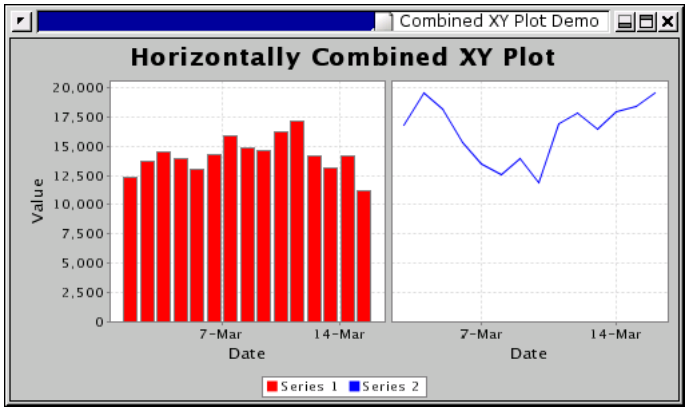
```
// return a new chart containing the overlaid plot...
return new JFreeChart("Overlaid Plot Example", JFreeChart.DEFAULT_TITLE_FONT, plot, true);
```

And that's all there is to it! The chart can be displayed in a `ChartPanel`, or used to draw into any `Graphics2D` instance.

9.3 Creating a CombinedXYPlot

9.3.1 Overview

A *combined XY plot* is a plot that displays two or more subplots (instances of `XYPlot`) sharing either the horizontal or the vertical axis. This section presents a sample application that displays a vertical XY bar chart and a line plot, side by side sharing the same range axis.



The procedure for creating this chart is fairly similar to that described in the previous section for the overlaid XY plot.

9.3.2 The Application

As in the previous example, the `TimeSeriesCollection` class is used to represent each dataset:

```
// create dataset 1...
TimeSeries series1 = new TimeSeries("Series 1", Day.class);
series1.add(new Day(1, SerialDate.MARCH, 2002), 12353.3);
series1.add(new Day(2, SerialDate.MARCH, 2002), 13734.4);
series1.add(new Day(3, SerialDate.MARCH, 2002), 14525.3);
series1.add(new Day(4, SerialDate.MARCH, 2002), 13984.3);
series1.add(new Day(5, SerialDate.MARCH, 2002), 12999.4);
series1.add(new Day(6, SerialDate.MARCH, 2002), 14274.3);
series1.add(new Day(7, SerialDate.MARCH, 2002), 15943.5);
series1.add(new Day(8, SerialDate.MARCH, 2002), 14845.3);
series1.add(new Day(9, SerialDate.MARCH, 2002), 14645.4);
series1.add(new Day(10, SerialDate.MARCH, 2002), 16234.6);
series1.add(new Day(11, SerialDate.MARCH, 2002), 17232.3);
series1.add(new Day(12, SerialDate.MARCH, 2002), 14232.2);
series1.add(new Day(13, SerialDate.MARCH, 2002), 13102.2);
series1.add(new Day(14, SerialDate.MARCH, 2002), 14230.2);
series1.add(new Day(15, SerialDate.MARCH, 2002), 11235.2);
TimeSeriesCollection collection = new TimeSeriesCollection(series1);

collection.setDomainIsPointsInTime(false);
// this tells the time series collection that
// we intend the data to represent time periods
// NOT points in time.  This is required when
// determining the min/max values in the
// dataset's domain.

return collection;
```

Similar code is used for the second dataset.

9.3.3 Constructing the Chart

Constructing the chart begins with the creation of the parent plot:

```
// create a parent plot...
CombinedXYPlot plot = new CombinedXYPlot(new VerticalNumberAxis("Value"),
                                         CombinedXYPlot.HORIZONTAL);
```

In this case, the layout is horizontal (plots side-by-side).

Next, the two sub-plots are created:

```
// create a drawing supplier to ensure all series use a unique paint/stroke...
DrawingSupplier supplier = new DefaultDrawingSupplier();

// create subplot 1...
IntervalXYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new VerticalXYBarRenderer(0.20);
renderer1.setDrawingSupplier(supplier);
renderer1.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy",
                                                             "0,000.0"));
XYPlot subplot1 = new XYPlot(data1,
                             new HorizontalDateAxis("Date"),
                             null,
                             renderer1);

// create subplot 2...
XYDataset data2 = this.createDataset2();
XYPlot subplot2 = new XYPlot(data2,
                             new HorizontalDateAxis("Date"),
                             null);
XYItemRenderer renderer2 = subplot2.getRenderer();
renderer2.setDrawingSupplier(supplier);
renderer2.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy",
                              "0,000.0"));

// add the subplots...
plot.add(subplot1, 1);
plot.add(subplot2, 1);
```

Notice how each of the subplots has a `null` domain axis, since they share the parent plot's domain axis.

You can control the amount of space allocated to each plot by specifying a *weight* for each plot as you add it to the parent plot. The weights are totalled, and each plot is allocated space based on its weight as a percentage of the total. In the example above, each plot is allocated the same weight (`1`) which means that each plot is drawn in half (`1/2`) the available space.

Finally, the chart is created:

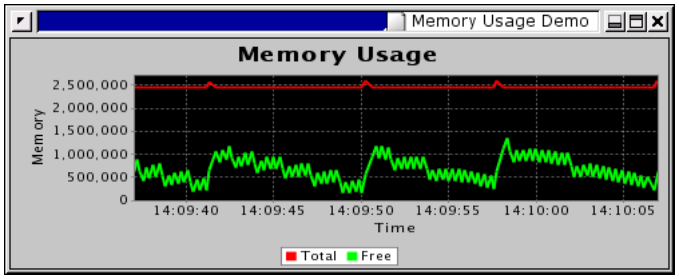
```
// return a new chart containing the overlaid plot...
return new JFreeChart("Horizontally Combined XY Plot",
                      JFreeChart.DEFAULT_TITLE_FONT, plot, true);
```

And that's it!

10 Dynamic Charts

10.1 Overview

To illustrate the use of JFreeChart for creating "dynamic" charts, this section presents a sample application that displays a frequently updating chart of JVM memory usage and availability.



10.2 Background

10.2.1 Event notification

JFreeChart uses an *event notification mechanism* that allows it to respond to changes to any component of the chart. Whenever a dataset is updated, a `DatasetChangeEvent` is sent to all registered listeners. Ultimately, this results in a `ChartChangeEvent` being raised—for charts that are displayed in a `ChartPanel`, this results in the panel redrawing the chart.

10.2.2 Performance

Regarding performance, you need to be aware that JFreeChart wasn't designed specifically for generating *real-time charts*. Each time a dataset is updated, the `ChartPanel` reacts by redrawing the entire chart. Optimisations, such as only drawing the most recently added data point, are difficult to implement in the general case, even more so given the `Graphics2D` abstraction (in the Java2D API) employed by JFreeChart. This limits the number of "frames per second" you will be able to achieve with JFreeChart. Whether this will be an issue for you depends on your data, the requirements of your application, and your operating environment. In other words, "your mileage may vary."

10.3 The Demo Application

10.3.1 Overview

The `MemoryUsage` demonstration is included in the "extra" download available to purchasers of this document.

10.3.2 Creating the Dataset

The dataset is created using two `TimeSeries` objects (one for the total memory and the other for the free memory) that are added to a single time series collection:

```
// create two series that automatically discard data > 30 seconds old...
this.total = new TimeSeries("Total", Millisecond.class);
this.total.setHistoryCount(30000);
this.free = new TimeSeries("Free", Millisecond.class);
this.free.setHistoryCount(30000);
TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(total);
dataset.addSeries(free);
```

The *history-count* attribute for each time series is set to 30,000 milliseconds (or 30 seconds) so that whenever new data is added to the series, any observations that are older that 30 seconds are automatically discarded.

10.3.3 Creating the Chart

The chart creation (and customisation) follows the standard pattern for all charts. No special steps are required to create a dynamic chart, except that you should ensure that the axes have their *auto-range* attribute set to `true`.

10.3.4 Updating the Dataset

In the demo, the dataset is updated by adding data to the two time series from a separate thread, managed by the following timer:

```
class DataGenerator extends Timer implements ActionListener {

    DataGenerator() {
        super(100, null);
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        long f = Runtime.getRuntime().freeMemory();
        long t = Runtime.getRuntime().totalMemory();
        addTotalObservation(t);
        addFreeObservation(f);
    }

}
```

Note that JFreeChart does not yet use thread synchronisation between the chart drawing code and the dataset update code, so this approach is a little unsafe. This will be addressed before version 1.0.0 is released.

One other point to note, at one point while investigating reports of a "memory leak" in JFreeChart, I left this demo running on a test machine for about six days. As the chart updates, you can see the effect of the garbage collector. Over the six day period, the total memory used remained constant while the free memory decreased as JFreeChart discarded temporary objects (garbage), and increased at the points where the garbage collector did its work.

For reference, here is the complete source code for the example:

```
package com.jrefinery.chart.demo;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.HorizontalDateAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.VerticalNumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.XYItemRenderer;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;

public class MemoryUsage extends JPanel {

    private TimeSeries total;

    private TimeSeries free;

    public MemoryUsage() {

        super(new BorderLayout());

        // create two series that automatically discard data more than 30 seconds old...
        this.total = new TimeSeries("Total", Millisecond.class);
        this.total.setHistoryCount(30000);
        this.free = new TimeSeries("Free", Millisecond.class);
        this.free.setHistoryCount(30000);
        TimeSeriesCollection dataset = new TimeSeriesCollection();
        dataset.addSeries(total);
        dataset.addSeries(free);

        HorizontalDateAxis domain = new HorizontalDateAxis("Time");
        VerticalNumberAxis range = new VerticalNumberAxis("Memory");

        XYPlot xyplot = new XYPlot(dataset, domain, range);
        xyplot.setBackgroundPaint(Color.black);
        XYItemRenderer renderer = xyplot.getRenderer();
        renderer.setSeriesPaint(0, Color.red);
        renderer.setSeriesPaint(1, Color.green);
        renderer.setDefaultStroke(new BasicStroke(2f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL));

        domain.setAutoRange(true);
        domain.setLowerMargin(0.0);
        domain.setUpperMargin(0.0);
        domain.setTickLabelsVisible(true);

        range.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

        JFreeChart chart = new JFreeChart("Memory Usage", JFreeChart.DEFAULT_TITLE_FONT,
                                          xyplot, true);
        ChartPanel chartPanel = new ChartPanel(chart);
        add(chartPanel);

    }

    private void addTotalObservation(double y) {
        total.add(new Millisecond(), y);
    }

    private void addFreeObservation(double y) {
        free.add(new Millisecond(), y);
    }

    class DataGenerator extends Timer implements ActionListener {

        DataGenerator() {
            super(100, null);
            addActionListener(this);
        }

        public void actionPerformed(ActionEvent event) {
```

```
            long f = Runtime.getRuntime().freeMemory();
            long t = Runtime.getRuntime().totalMemory();
            addTotalObservation(t);
            addFreeObservation(f);
        }

    }

    public static void main(String[] args) {

        JFrame frame = new JFrame("Memory Usage Demo");
        MemoryUsage panel = new MemoryUsage();
        frame.getContentPane().add(panel, BorderLayout.CENTER);
        frame.setBounds(200, 120, 500, 200);
        frame.setVisible(true);
        panel.new DataGenerator().start();

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

}
```

11 Tooltips

11.1 Overview

JFreeChart includes mechanisms for generating, collecting and displaying tool tips for individual components of a chart.

In this section, I describe:

- how to generate tool tips (including customisation of tool tips);
- how tool tips are collected;
- how to display tool tips;
- how to disable tool tips if you don't need them;

11.2 Generating Tool Tips

If you want to use tool tips, you need to make sure they are generated as your chart is being drawn. You do this by setting a tool tip generator for your plot or, in many cases, the plot's item renderer.

In the sub-sections that follow, I describe how to set a tool tip generator for the common chart types.

11.2.1 Pie Charts

The `PiePlot` class generates tool tips using the `PieToolTipGenerator` interface. A standard implementation (`StandardPieToolTipGenerator`) is provided, and you are free to create your own implementations.

To set the tool tip generator, use the following method in the `PiePlot` class:

> `public void setToolTipGenerator(PieToolTipGenerator generator);`
> Sets the tool tip generator for the pie chart. If you set this to `null`, no tool tips will be generated.

11.2.2 Category Charts

Category charts—including most of the bar charts generated by JFreeChart—are based on the `CategoryPlot` class and use a `CategoryItemRenderer` to draw each data item. The renderer generates tool tips (if required) using a `CategoryToolTipGenerator`.

To set the tool tip generator for a category plot's item renderer, use the following method (defined in the `AbstractCategoryItemRenderer` class):

> `public void setToolTipGenerator(CategoryToolTipGenerator generator);`
> Sets the tool tip generator for the renderer. If you set this to `null`, no tool tips will be generated.

11.2.3 XY Charts

XY charts—including scatter plots and all the time series charts generated by JFreeChart—are based on the `XYPlot` class and use an `XYItemRenderer` to draw each data item. The renderer generates tool tips (if required) using an `XYToolTipGenerator`.

To set the tool tip generator for an XY plot's item renderer, use the following method (defined in the `AbstractXYItemRenderer` class):

> `public void setToolTipGenerator(XYToolTipGenerator generator);`
> Sets the tool tip generator for the renderer. If you set this to `null`, no tool tips will be generated.

11.3 Collecting Tool Tips

Tool tips are collected, along with other chart entity information, using the `ChartRenderingInfo` class. You need to supply an instance of this class to `JFreeChart`'s `draw(...)` method, otherwise no tool tip information will be recorded (even if a generator has been registered with the plot or the plot's item renderer, as described in the previous sections).

Fortunately, the `ChartPanel` class takes care of this automatically, so if you are displaying your charts using the `ChartPanel` class you do not need to worry about how tool tips are collected—it is done for you.

11.4 Displaying Tool Tips

Tool tips are automatically displayed by the `ChartPanel` class, provided that you have set up a tool tip generator for the plot (or the plot's renderer).

You can also enable or disable the *display* of tool tips in the `ChartPanel` class, using this method:

> `public void setDisplayToolTips(boolean flag);`
> Switches the display of tool tips on or off.

11.5 Disabling Tool Tips

The most effective way to disable tool tips is to set the tool tip generator to `null`. This ensures that no tool tip information is even generated, which can save memory and processing time (particularly for charts with large datasets).

You can also disable the *display* of tool tips in the `ChartPanel` class, using the method given in the previous section.

11.6 Customising Tool Tips

You can take full control of the text generated for each tool tip by providing your own implementation of the appropriate tool tip generator interface.

```
X           | SERIES1 | SERIES2 | SERIES3
------------+---------+---------+--------
1-Aug-2002 |    54.3 |    32.1 |    53.4
2-Aug-2002 |    43.4 |    54.3 |    75.2
3-Aug-2002 |    39.6 |    55.9 |    37.1
4-Aug-2002 |    35.4 |    55.2 |    27.5
5-Aug-2002 |    33.9 |    49.8 |    22.3
6-Aug-2002 |    35.2 |    48.4 |    17.7
7-Aug-2002 |    38.9 |    49.7 |    15.3
8-Aug-2002 |    36.3 |    44.4 |    12.1
9-Aug-2002 |    31.0 |    46.3 |    11.0
```

You should set up a test database containing these tables...ask your database administrator to help you if necessary. I've called my test database `jfreechartdb`, but you can change the name if you want to.

In the next section I document the steps I used to set up this sample data using*PostgreSQL*, the database system that I have available for testing purposes. If you are using a different system, you may need to perform a slightly different procedure—refer to your database documentation for information.

12.4 PostgreSQL

12.4.1 About PostgreSQL

PostgreSQL is a powerful object-relational database server, distributed under an open-source licence. You can find out more about PostgreSQL at:

```
http://www.postgresql.org
```

Note: although PostgreSQL is free, it has most of the features of large commercial relational database systems. I encourage you to install it and try it out.

12.4.2 Creating a New Database

First, while logged in as the database administrator, I create a test database called `jfreechartdb`:

```
CREATE DATABASE jfreechartdb;
```

Next, I create a user `jfreechart`:

```
CREATE USER jfreechart WITH PASSWORD 'password';
```

This username and password will be used to connect to the database via JDBC.

12.4.3 Creating the Pie Chart Data

To create the table for the pie dataset:

```
CREATE TABLE piedata1 (
    category VARCHAR(32),
    value    FLOAT
);
```

...and to populate it:

```
INSERT INTO piedata1 VALUES ('London',   54.3);
INSERT INTO piedata1 VALUES ('New York', 43.4);
INSERT INTO piedata1 VALUES ('Paris',    17.9);
```

12.4.4 Creating the Category Chart Data

To cr043834u96305(17.9)(Categor)1(y)-384(C)1(hart)-383(Data91sXs)c7.9he)4(C)1(har383(VALs9he10010-2
INSE9T INTO piedata1 y(VALUES)-525(('London)-1(Nhe)-1(rk',)-43.4);

12.5 The JDBC Driver

To access the sample data via JDBC, you need to obtain a JDBC driver for your database. For PostgreSQL, I downloaded a free driver from:

`http://jdbc.postgresql.org`

In order to use this driver, I need to ensure that the jar file containing the driver is on the classpath.

12.6 The Demo Applications

12.6.1 JDBCPieChartDemo

The `JDBCPieChartDemo` application will generate a pie chart using the data in the `piedata1` table, providing that you have configured your database correctly.

The code for reading the data is in the `readData()` method:

```
private PieDataset readData() {

    JDBCPieDataset data = null;

    String url = "jdbc:postgresql://nomad/jfreechartdb";
    Connection con;

    try {
        Class.forName("org.postgresql.Driver");
    }
    catch (ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url, "jfreechart", "password");

        data = new JDBCPieDataset(con);
        String sql = "SELECT * FROM PIEDATA1;";
        data.executeQuery(sql);
        con.close();
    }

    catch (SQLException e) {
        System.err.print("SQLException: ");
        System.err.println(e.getMessage());
    }

    catch (Exception e) {
        System.err.print("Exception: ");
        System.err.println(e.getMessage());
    }

    return data;

}
```

Important things to note in the code are:

- the `url` used to reference the test database includes the name of my test server (`nomad`), you will need to modify this;
- a connection is made to the database using the username/password combination `jfreechart/password`;

- the query used to pull the data from the database is a standard SELECT query, but you can use any SQL query as long as it returns columns in the required format (refer to the `JDBCPieDataset` class documentation for details).

12.6.2 JDBCCategoryChartDemo

The `JDBCCategoryChartDemo` application generates a bar chart using the data in the `categorydata1` table. The code is almost identical to the `JDBCPieChartDemo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the `JDBCCategoryDataset` class documentation for details).

12.6.3 JDBCXYChartDemo

The `JDBCXYChartDemo` application generates a time series chart using the data in the `xydata1` table. The code is almost identical to the `JDBCPieChartDemo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the `JDBCXYDataset` class documentation for details).

13 Exporting Charts to Acrobat PDF

13.1 Introduction

In this section, I describe how to export a chart to an Acrobat PDF file using JFreeChart and iText. Along with the description, I provide a small demonstration application that creates a PDF file containing a basic chart. The resulting file can be viewed using Acrobat Reader, or any other software that is capable of reading and displaying PDF files.

13.2 What is Acrobat PDF?

Acrobat PDF is a widely used electronic document format. Its popularity is due, at least in part, to its ability to reproduce high quality output on a variety of different platforms.

PDF was created by Adobe Systems Incorporated. Adobe provide a free (but closed source) application called *Acrobat Reader* for reading PDF documents. Acrobat Reader is available on most end-user computing platforms, including GNU/Linux, Windows, Unix, Macintosh and others.

If your system doesn't have Acrobat Reader installed, you can download a copy from:

```
http://www.adobe.com/products/acrobat/readstep.html
```

On some platforms, there are free (in the GNU sense) software packages available for viewing PDF files. Ghostview on Linux is one example.

13.3 iText

iText is a popular free Java class library for creating documents in PDF format. It is developed by Bruno Lowagie, Paulo Soares and others.

The home page for iText is:

```
http://www.lowagie.com/iText
```

At the time of writing, the latest version of iText is `0.98`.

13.4 Graphics2D

JFreeChart can work easily with iText because iText provides a `Graphics2D` implementation. Before I proceed to the demonstration application, I will briefly review the `Graphics2D` class.

The `java.awt.Graphics2D` class, part of the standard Java 2D API, defines a range of methods for drawing text and graphics in a two dimensional space. Particular subclasses of `Graphics2D` handle all the details of mapping the output (text and graphics) to specific devices.

JFreeChart has been designed to draw charts using only the methods defined by the `Graphics2D` class. This means that JFreeChart can generate output to any target that can provide a `Graphics2D` subclass.

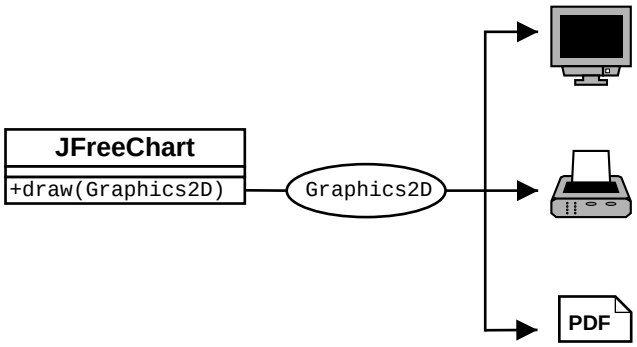


Figure 1: The JFreeChart `draw(...)` method

iText incorporates a `PdfGraphics2D` class, which means that iText is capable of generating PDF content based on calls to the methods defined by the `Graphics2D` class...and this makes it easy to produce charts in PDF format, as you will see in the following sections.

13.5 Getting Started

To compile and run the demonstration application, you will need the following jar files:

File:	Description:
`jfreechart-0.9.8.jar`	The JFreeChart class library.
`jcommon-0.8.0.jar`	The JCommon class library (used by JFreeChart).
`iText-0.98.jar`	The iText class library.

The first two files are included with JFreeChart, and the fourth is the iText runtime.

13.6 The Application

The first thing the sample application needs to do is create a chart. I've copied the time series chart from the `TimeSeriesDemo` class:

```
// create a chart...
String chartTitle = "Legal & General Unit Trust Prices";
XYDataset dataset = createDataset();

JFreeChart chart = ChartFactory.createTimeSeriesChart(
    chartTitle,
    "Date", "Price Per Unit",
    dataset,
    true,
    true,
    false
);

// some additional chart customisation here...
```

There is nothing special here—in fact you could replace the code above with any other code that creates a `JFreeChart` object. You are encouraged to experiment.

Next, I will save a copy of the chart in a PDF file:

```
// write the chart to a PDF file...
File fileName = new File(System.getProperty("user.home") + "/jfreechart1.pdf");
saveChartAsPDF(fileName, chart, 400, 300, new DefaultFontMapper());
```

There are a couple of things to note here.

First, I have hard-coded the filename used for the PDF file. I've done this to keep the sample code short. In a real application, you would provide some other means for the user to specify the filename, perhaps by presenting a file chooser dialog.

Second, the `saveChartAsPDF(...)` method hasn't been implemented yet! To create that method, I'll first write another more general method, `writeChartAsPDF(...)`. This method performs most of the work that will be required by the `saveChartAsPDF(...)` method, but it writes data to an *output stream* rather than a file.

```
public static void writeChartAsPDF(OutputStream out,
                                   JFreeChart chart,
                                   int width, int height,
                                   FontMapper mapper) throws IOException {

    Rectangle pagesize = new Rectangle(width, height);
    Document document = new Document(pagesize, 50, 50, 50, 50);

    try {
        PdfWriter writer = PdfWriter.getInstance(document, out);

        document.addAuthor("JFreeChart");
        document.addSubject("Demonstration");
        document.open();

        PdfContentByte cb = writer.getDirectContent();
        PdfTemplate tp = cb.createTemplate(width, height);
        Graphics2D g2 = tp.createGraphics(width, height, mapper);

        Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
        chart.draw(g2, r2D, null);
        g2.dispose();
        cb.addTemplate(tp, 0, 0);
    }
    catch(DocumentException de) {
        System.err.println(de.getMessage());
    }

    document.close();

}
```

Inside this method, you will see some code that sets up and opens an iText document, obtains a `Graphics2D` instance from the document, draws the chart using the `Graphics2D` object, and closes the document.

You will also notice that one of the parameters for this method is a `FontMapper` object. The `FontMapper` interface maps Java `Font` objects to the `BaseFont` objects used by iText.

The `DefaultFontMapper` class is predefined with default mappings for the Java *logical fonts*. If you use only these fonts, then it is enough to create a `DefaultFontMapper` using the default constructor. If you want to use other fonts (for example, a font that supports a particular character set) then you need to do more work. I'll give an example of this later.

In the implementation of the `writeChartAsPDF(...)` method, I've chosen to create a PDF document with a custom page size (matching the requested size

of the chart). You can easily adapt the code to use a different page size, alter the size and position of the chart and even draw multiple charts inside one PDF document.

Now that I have a method to send PDF data to an output stream, it is straightforward to implement the `saveChartAsPDF(...)` method. Simply create a `FileOutputStream` and pass it on to the `writeChartAsPDF(...)` method:

```
public static void saveChartAsPDF(File file,
                                  JFreeChart chart,
                                  int width, int height,
                                  FontMapper mapper) throws IOException {

    OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
    writeChartAsPDF(out, chart, width, height, mapper);
    out.close();

}
```

This is all the code that is required. The pieces can be assembled into the following program (reproduced in full here so that you can see all the required import statements and the context in which the code is run):

```
package com.jrefinery.chart.demo;

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.text.SimpleDateFormat;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.StandardLegend;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.StandardXYItemRenderer;
import org.jfree.chart.renderer.XYItemRenderer;
import org.jfree.data.XYDataset;
import org.jfree.data.time.Month;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;

import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Rectangle;
import com.lowagie.text.pdf.DefaultFontMapper;
import com.lowagie.text.pdf.FontMapper;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfTemplate;
import com.lowagie.text.pdf.PdfWriter;

/**
 * A simple demonstration showing how to write a chart to PDF format using
 * JFreeChart and iText.
 * <P>
 * You can download iText from http://www.lowagie.com/iText.
 */
public class ChartToPDFDemo {

    /**
     * Saves a chart to a PDF file.
     * @param file The file.
     * @param chart The chart.
     * @param width The chart width.
     * @param height The chart height.
     */
```

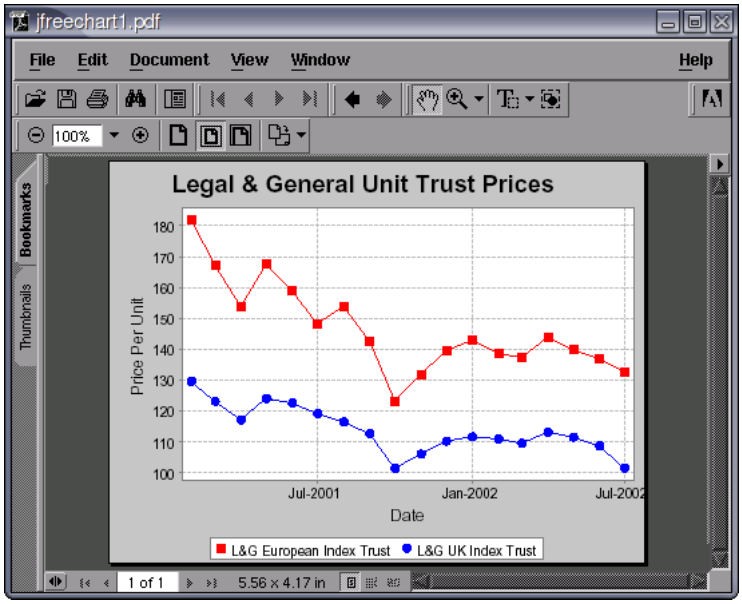
```
public static void saveChartAsPDF(
    File file,
    JFreeChart chart,
    int width,
    int height,
    FontMapper mapper)
    throws IOException {

    OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
    writeChartAsPDF(out, chart, width, height, mapper);
    out.close();

}

/**
 * Writes a chart to an output stream in PDF format.
 *
 * @param out The output stream.
 * @param chart The chart.
 * @param width The chart width.
 * @param height The chart height.
 */
public static void writeChartAsPDF(
    OutputStream out,
    JFreeChart chart,
    int width,
    int height,
    FontMapper mapper)
    throws IOException {

    Rectangle pagesize = new Rectangle(width, height);
    Document document = new Document(pagesize, 50, 50, 50, 50);
    try {
        PdfWriter writer = PdfWriter.getInstance(document, out);
        document.addAuthor("JFreeChart");
        document.addSubject("Demonstration");
        document.open();
        PdfContentByte cb = writer.getDirectContent();
        PdfTemplate tp = cb.createTemplate(width, height);
        Graphics2D g2 = tp.createGraphics(width, height, mapper);
        Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
        chart.draw(g2, r2D, null);
        g2.dispose();
        cb.addTemplate(tp, 0, 0);
    } catch (DocumentException de) {
        System.err.println(de.getMessage());
    }
    document.close();
}

/**
 * Creates a dataset, consisting of two series of monthly data.
 *
 * @return the dataset.
 */
public static XYDataset createDataset() {

    TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
    s1.add(new Month(2, 2001), 181.8);
    s1.add(new Month(3, 2001), 167.3);
    s1.add(new Month(4, 2001), 153.8);
    s1.add(new Month(5, 2001), 167.6);
    s1.add(new Month(6, 2001), 158.8);
    s1.add(new Month(7, 2001), 148.3);
    s1.add(new Month(8, 2001), 153.9);
    s1.add(new Month(9, 2001), 142.7);
    s1.add(new Month(10, 2001), 123.2);
    s1.add(new Month(11, 2001), 131.8);
    s1.add(new Month(12, 2001), 139.6);
    s1.add(new Month(1, 2002), 142.9);
    s1.add(new Month(2, 2002), 138.7);
    s1.add(new Month(3, 2002), 137.3);
    s1.add(new Month(4, 2002), 143.9);
    s1.add(new Month(5, 2002), 139.8);
```

```
        s1.add(new Month(6, 2002), 137.0);
        s1.add(new Month(7, 2002), 132.8);

        TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
        s2.add(new Month(2, 2001), 129.6);
        s2.add(new Month(3, 2001), 123.2);
        s2.add(new Month(4, 2001), 117.2);
        s2.add(new Month(5, 2001), 124.1);
        s2.add(new Month(6, 2001), 122.6);
        s2.add(new Month(7, 2001), 119.2);
        s2.add(new Month(8, 2001), 116.5);
        s2.add(new Month(9, 2001), 112.7);
        s2.add(new Month(10, 2001), 101.5);
        s2.add(new Month(11, 2001), 106.1);
        s2.add(new Month(12, 2001), 110.3);
        s2.add(new Month(1, 2002), 111.7);
        s2.add(new Month(2, 2002), 111.0);
        s2.add(new Month(3, 2002), 109.6);
        s2.add(new Month(4, 2002), 113.2);
        s2.add(new Month(5, 2002), 111.6);
        s2.add(new Month(6, 2002), 108.8);
        s2.add(new Month(7, 2002), 101.6);

        TimeSeriesCollection dataset = new TimeSeriesCollection();
        dataset.addSeries(s1);
        dataset.addSeries(s2);

        return dataset;

    }

    /**
     * Starting point for the demonstration application.
     */
    public static void main(String[] args) {
        try {
            // create a chart...
            String chartTitle = "Legal & General Unit Trust Prices";
            XYDataset dataset = createDataset();

            JFreeChart chart =
                ChartFactory.createTimeSeriesChart(
                    chartTitle,
                    "Date",
                    "Price Per Unit",
                    dataset,
                    true,
                    true,
                    false);

            StandardLegend sl = (StandardLegend) chart.getLegend();
            sl.setDisplaySeriesShapes(true);

            XYPlot plot = chart.getXYPlot();
            XYItemRenderer renderer = plot.getRenderer();
            if (renderer instanceof StandardXYItemRenderer) {
                StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
                rr.setPlotShapes(true);
                rr.setDefaultShapeFilled(true);
            }
            DateAxis axis = (DateAxis) plot.getDomainAxis();
            axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));

            // write the chart to a PDF file...
            File fileName =
                new File(System.getProperty("user.home") + "/jfreechart1.pdf");
            saveChartAsPDF(fileName, chart, 400, 300, new DefaultFontMapper());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Before you compile and run the application, remember to change the file name used for the PDF file to something appropriate for your system! And include the jar files listed in section 13.5 on your classpath.

13.7 Viewing the PDF File

After compiling and running the sample application, you can view the resulting PDF file using Acrobat Reader:



Acrobat Reader provides a zooming facility to allow you to get a close up view of your charts.

13.8 Unicode Characters

It is possible to use the full range of Unicode characters in JFreeChart and iText, as long as you are careful about which fonts you use. In this section, I present some modifications to the previous example to show how to do this.

13.8.1 Background

Internally, Java uses the Unicode character encoding to represent text strings. This encoding uses sixteen bits per character, which means there are potentially 65,536 different characters available (the Unicode standard defines something like 38,000 characters).

You can use any of these characters in both JFreeChart and iText, subject to one proviso: *the font you use to display the text must define the characters used or you will not be able to see them.*

Many fonts are not designed to display the entire Unicode character set. The following website contains useful information about fonts that do support Unicode (at least to some extent):

```
http://www.ccss.de/slovo/unifonts.htm
```

I have tried out the `Arial Unicode MS` font with success—in fact, I will use this font in the example that follows. But you should bear in mind that supporting the full Unicode character set means that the font definition file is quite large: the `arialuni.ttf` file weighs in at `24,131,012` bytes on my system.

13.8.2 Fonts, iText and Java

iText has to handle fonts according to the PDF specification. This deals with document portability by allowing fonts to be (optionally) embedded in a PDF file. This requires access to the font definition file.

Java, on the other hand, abstracts away some of the details of particular font formats with the use of the `Font` class.

To support the `Graphics2D` implementation in iText, it is necessary to map `Font` objects from Java to `BaseFont` objects in iText. This is the role of the `FontMapper` interface.

If you create a new `DefaultFontMapper` instance using the default constructor, it will already contain sensible mappings for the logical fonts defined by the Java specification. But if you want to use additional fonts—and you must if you want to use a wide range of Unicode characters—then you need to add extra mappings to the `DefaultFontMapper` object.

13.8.3 Mapping Additional Fonts

I've decided to use the `Arial Unicode MS` font to display a chart title that incorporates some Unicode characters. The font definition file (`arialuni.ttf`) is located, on my system, in the directory:

```
/usr/lib/java2/jre/lib/fonts
```

Here's the code used to create the `FontMapper` for use by iText—I've based this on an example written by Paulo Soares:

```
DefaultFontMapper mapper = new DefaultFontMapper();
mapper.insertDirectory("/usr/lib/java2/jre/lib/fonts");
DefaultFontMapper.BaseFontParameters pp =
                    mapper.getBaseFontParameters("Arial Unicode MS");
if (pp!=null) {
    pp.encoding = BaseFont.IDENTITY_H;
}
```

Now I can modify the code that creates the chart, in order to add a custom title to the chart (I've changed the data and chart type also):

```
// create a chart...
TimeSeries series = new TimeSeries("Random Data");

Day current = new Day(1, 1, 2000);
double value = 100.0;
for (int i = 0; i < 1000; i++) {
    try {
        value = value + Math.random() - 0.5;
        series.add(current, new Double(value));
        current = (Day) current.next();
    }
    catch (SeriesException e) {
```

```
            System.err.println("Error adding to series");
        }
    }

    XYDataset data = new TimeSeriesCollection(series);

    JFreeChart chart = ChartFactory.createTimeSeriesChart(
        "Test", "Date", "Value", data, true, false, false
    );

    // Unicode test...
    String text = "\u278A\u20A0\u20A1\u20A2\u20A3\u20A4\u20A5\u20A6\u20A7\u20A8\u20A9";
    Font font = new Font("Arial Unicode MS", Font.PLAIN, 12);
    TextTitle subtitle = new TextTitle(text, font);
    chart.addSubtitle(subtitle);
```

Notice that the subtitle (which mostly consists of a meaningless collection of currency symbols) is defined using escape sequences to specify each Unicode character. This avoids any problems with encoding conversions when I save the Java source file.

The output from the modified sample program is shown in figure 2. The example has been embedded in this document in PDF format, so it is a good example of the type of output you can expect by following the instructions in this document.

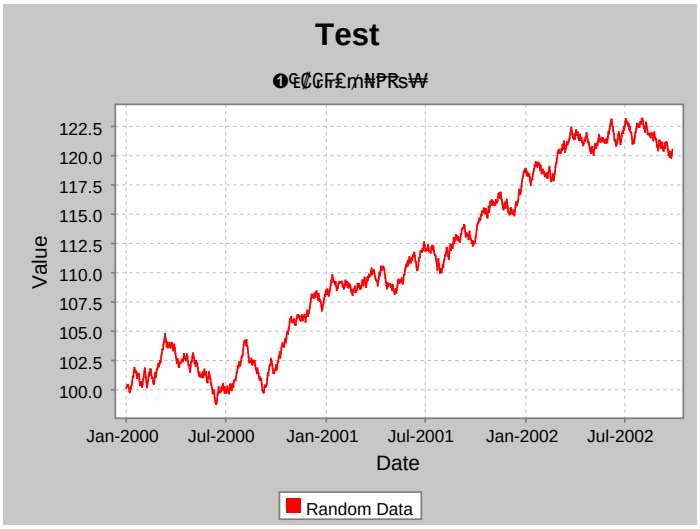


Figure 2: A Unicode subtitle

14 Exporting Charts to SVG Format

14.1 Introduction

In this section, I present an example that shows how to export charts to SVG format, using JFreeChart and Batik (an open source library for working with SVG).

14.2 Background

14.2.1 What is SVG?

Scalable Vector Graphics (SVG) is a standard language for describing two-dimensional graphics in XML format. It is a *Recommendation* of the World Wide Web Consortium (W3C).

14.2.2 Batik

Batik is an open source toolkit, written in Java, that allows you to generate SVG content. Batik is available from:

```
http://xml.apache.org/batik
```

At the time of writing, the latest *stable* version of Batik is `1.1.1`. However, for the demonstration below, I have used the `1.5beta4b` release.

14.3 A Sample Application

14.3.1 JFreeChart and Batik

JFreeChart and Batik can work together relatively easily because:

- JFreeChart draws all chart output using Java's `Graphics2D` abstraction; and
- Batik provides a concrete implementation of `Graphics2D` that generates SVG output (`SVGGraphics2D`).

In this section, a simple example is presented to get you started using JFreeChart and Batik.

14.3.2 Getting Started

First, you should download Batik and install it according to the instructions provided on the Batik web page.

To compile and run the sample program presented in the next section, you need to ensure that the following jar files are on your classpath:

File:	Description:
jcommon-0.8.0.jar	Common classes from The Object Refinery.
jfreechart-0.9.8.jar	The JFreeChart class library.
batik-awt-util.jar	Batik runtime files.
batik-dom.jar	Batik runtime files.
batik-ext.jar	Batik runtime files.
batik-svggen.jar	Batik runtime files.
batik-util.jar	Batik runtime files.
batik-xml.jar	Batik runtime files.

14.3.3 The Application

Create a project in your favourite Java development environment, add the libraries listed in the previous section, and type in the following program:

```
package com.jrefinery.chart.premium.demo;

import java.awt.geom.Rectangle2D;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

import org.apache.batik.dom.GenericDOMImplementation;
import org.apache.batik.svggen.SVGGraphics2D;
import org.w3c.dom.DOMImplementation;
import org.w3c.dom.Document;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.DefaultPieDataset;

/**
 * A demonstration showing the export of a chart to SVG format.
 *
 * @author David Gilbert
 */
public class SVGExportDemo {

    /**
     * Starting point for the demo.
     *
     * @param args  ignored.
     */
    public static void main(String[] args) throws IOException {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();
        data.setValue("Category 1", new Double(43.2));
        data.setValue("Category 2", new Double(27.9));
        data.setValue("Category 3", new Double(79.5));

        // create a chart
        JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart", data, true,
                                                       false, false);

        // THE FOLLOWING CODE BASED ON THE EXAMPLE IN THE BATIK DOCUMENTATION...

        // Get a DOMImplementation
        DOMImplementation domImpl = GenericDOMImplementation.getDOMImplementation();

        // Create an instance of org.w3c.dom.Document
        Document document = domImpl.createDocument(null, "svg", null);

        // Create an instance of the SVG Generator
        SVGGraphics2D svgGenerator = new SVGGraphics2D(document);

        // Ask the chart to render into the SVG Graphics2D implementation
        chart.draw(svgGenerator, new Rectangle2D.Double(0, 0, 400, 300), null);
```

```
            // Finally, stream out SVG to a file using UTF-8 character to byte encoding
            boolean useCSS = true;
            Writer out = new OutputStreamWriter(
                new FileOutputStream(new File("test.svg")), "UTF-8");
            svgGenerator.stream(out, useCSS);

        }

    }
```

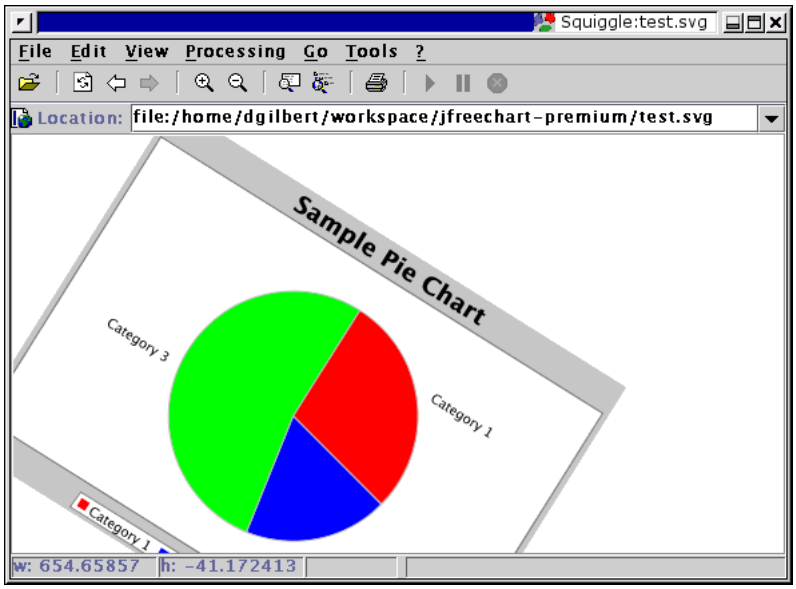
Running this program creates a file `test.svg` in SVG format.

14.3.4 Viewing the SVG

Batik includes a viewer application ("Squiggle") which you can use to open and view the SVG file. The Batik download includes instructions for running the viewer, effectively all you require is:

```
java -jar batik.jar
```

The following screen shot shows the pie chart that we created earlier, displayed using the browser application. A transformation (rotation) has been applied to the chart from within the browser:



If you play about with the viewer, zooming in and out and applying various transformations to the chart, you will begin to appreciate the power of the SVG format.

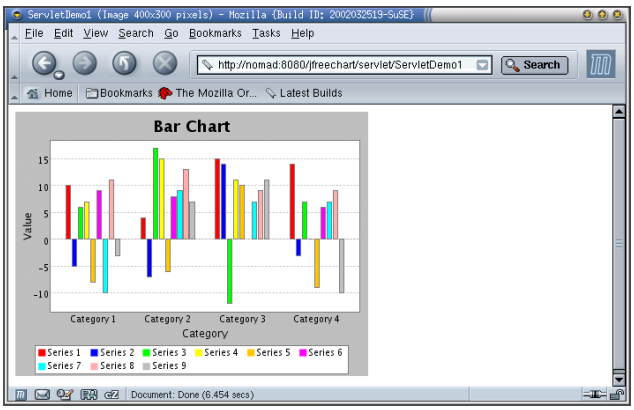
15 Servlets

15.1 Introduction

Sun's *Java Servlet* specification is a very popular technology for developing web applications. JFreeChart is well suited for use in a servlet environment and, in this section, a couple of basic examples are presented.

15.2 A Simple Servlet

The `ServletDemo1` class, included in the JFreeChart "premium demo" distribution, implements a very simple servlet that returns a PNG image of a bar chart generated using JFreeChart. When it is run, the browser will display a raw image (without any surrounding HTML), like this:



Typically, you will not present raw output in this way, so this servlet is not especially useful on its own. But it does illustrate the *request-response* nature of servlets, and is useful as a test case if you are configuring a server environment and want to check that everything is working. A second example, presented later, adds some surrounding HTML to the chart.

Here is the code for the basic servlet (stripped of comments):

```
package com.jrefinery.chart.premium.demo;

import java.io.OutputStream;
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.ChartUtilities;
import com.jrefinery.data.CategoryDataset;
import com.jrefinery.data.DatasetUtilities;
import com.jrefinery.data.DefaultCategoryDataset;

/**
 * A basic servlet that returns a PNG image file generated by JFreeChart.
 * <P>
 * This class is described in the JFreeChart Developer Guide.
 *
 * @author David Gilbert
```

```
 */
public class ServletDemo1 extends HttpServlet {

    public ServletDemo1() {
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        OutputStream out = response.getOutputStream();
        try {
            DefaultCategoryDataset dataset = new DefaultCategoryDataset();
            dataset.addValue(10.0, "S1", "C1");
            dataset.addValue(4.0, "S1", "C2");
            dataset.addValue(15.0, "S1", "C3");
            dataset.addValue(14.0, "S1", "C4");
            dataset.addValue(-5.0, "S2", "C1");
            dataset.addValue(-7.0, "S2", "C2");
            dataset.addValue(14.0, "S2", "C3");
            dataset.addValue(-3.0, "S2", "C4");
            dataset.addValue(6.0, "S3", "C1");
            dataset.addValue(17.0, "S3", "C2");
            dataset.addValue(-12.0, "S3", "C3");
            dataset.addValue( 7.0, "S3", "C4");
            dataset.addValue(7.0, "S4", "C1");
            dataset.addValue(15.0, "S4", "C2");
            dataset.addValue(11.0, "S4", "C3");
            dataset.addValue(0.0, "S4", "C4");
            dataset.addValue(-8.0, "S5", "C1");
            dataset.addValue(-6.0, "S5", "C2");
            dataset.addValue(10.0, "S5", "C3");
            dataset.addValue(-9.0, "S5", "C4");
            dataset.addValue(9.0, "S6", "C1");
            dataset.addValue(8.0, "S6", "C2");
            dataset.addValue(null, "S6", "C3");
            dataset.addValue(6.0, "S6", "C4");
            dataset.addValue(-10.0, "S7", "C1");
            dataset.addValue(9.0, "S7", "C2");
            dataset.addValue(7.0, "S7", "C3");
            dataset.addValue(7.0, "S7", "C4");
            dataset.addValue(11.0, "S8", "C1");
            dataset.addValue(13.0, "S8", "C2");
            dataset.addValue(9.0, "S8", "C3");
            dataset.addValue(9.0, "S8", "C4");
            dataset.addValue(-3.0, "S9", "C1");
            dataset.addValue(7.0, "S9", "C2");
            dataset.addValue(11.0, "S9", "C3");
            dataset.addValue(-10.0, "S9", "C4");

            JFreeChart chart = ChartFactory.createVerticalBarChart(
                                "Bar Chart",
                                "Category",
                                "Value",
                                dataset,
                                true, true, false
                               );
            response.setContentType("image/png");
            ChartUtilities.writeChartAsPNG(out, chart, 400, 300);
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
        finally {
            out.close();
        }

    }

}
```

The `doGet(...)` method is called by the servlet engine when a request is made by a client (usually a web browser). In response to the request, the servlet performs several steps:

- an `OutputStream` reference is obtained for returning output to the client;
- a chart is created;
- the *content type* for the response is set to `image/png`. This tells the client what type of data it is receiving;
- a PNG image of the chart is written to the output stream;
- the output stream is closed.

Note that the classes in the `javax.servlet.*` package (and sub-packages), used by the demo servlet, are not part of the *Java 2 Standard Edition (J2SE)*. In order to compile the above code using J2SE, you will need to obtain a `servlet.jar` file...I've used the one that is redistributed with Tomcat (an open source servlet engine written using Java):

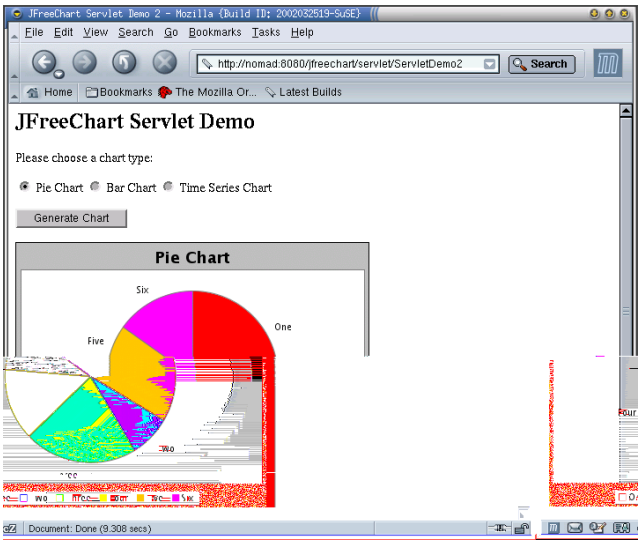
```
http://jakarta.apache.org/tomcat
```

The steps required to deploy this servlet to a servlet engine will be described later in this section. The examples in this section work well with Tomcat, but should also work with other servlet engines too (because servlets are designed to be portable between servlet engines).

15.3 Embedding Charts in HTML Pages

It is possible to embed a chart image generated by a servlet inside an HTML page generated by another servlet. This is demonstrated by `ServletDemo2`, which is also included in the JFreeChart "premium demo" distribution.

`ServletDemo2` processes a request by returning a page of HTML that, in turn, references another servlet (`ServletDemo2ChartGenerator`) that returns a PNG image of a chart. The end result is a chart embedded in an HTML page, like this:



Here is the code for `ServletDemo2`:

```
package com.jrefinery.chart.premium.demo;

import java.io.PrintWriter;
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;

public class ServletDemo2 extends HttpServlet {

    public ServletDemo2() {
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = new PrintWriter(response.getWriter());
        try {

            String param = request.getParameter("chart");

            response.setContentType("text/html");
            out.println("<HTML>");
            out.println("<HEAD>");
            out.println("<TITLE>JFreeChart Servlet Demo 2</TITLE>");
            out.println("</HEAD>");
            out.println("<BODY>");
            out.println("<H2>JFreeChart Servlet Demo</H2>");
            out.println("<P>");
            out.println("Please choose a chart type:");

            out.println("<FORM ACTION=\"ServletDemo2\" METHOD=POST>");
            String pieChecked = (param.equals("pie") ? " CHECKED" : "");
            String barChecked = (param.equals("bar") ? " CHECKED" : "");
            String timeChecked = (param.equals("time") ? " CHECKED" : "");
            out.println("<INPUT TYPE=\"radio\" NAME=\"chart\"
            VALUE=\"pie\"" + pieChecked + "> Pie Chart");
            out.println("<INPUT TYPE=\"radio\" NAME=\"chart\"
            VALUE=\"bar\"" + barChecked + "> Bar Chart");
            out.println("<INPUT TYPE=\"radio\" NAME=\"chart\"
            VALUE=\"time\"" + timeChecked + "> Time Series Chart");
            out.println("<P>");
            out.println("<INPUT TYPE=\"submit\" VALUE=\"Generate Chart\">");
            out.println("</FORM>");

            out.println("<P>");
            out.println("<IMG SRC=\"ServletDemo2ChartGenerator?type=" + param
                + "\" BORDER=1 WIDTH=400 HEIGHT=300/>");
            out.println("</BODY>");
            out.println("</HTML>");
            out.flush();
            out.close();
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
        finally {
            out.close();
        }

    }

}
```

Notice how this code gets a reference to a `Writer` from the `response` parameter, rather than an `OutputStream` as in the previous example. The reason for this is because this servlet will be returning text (HTML), compared to the previous servlet which returned binary data (a PNG image).[5]

[5]The `Writer` is wrapped in a `PrintWriter` in order to use the more convenient methods

The response type is set to `text/html` since this servlet returns HTML text. An important point to note is that the `<IMG>` tag in the HTML references another servlet (`ServletDemo2ChartGenerator`), and this other servlet creates the required chart image. The actual chart returned is controlled by the `chart` parameter, which is set up in the HTML using a `<FORM>` element.

15.4 Supporting Files

Servlets typically generate output for clients that access the web application via a web browser. Most web applications will include at least one HTML page that is used as the starting point for the application.

For the demo servlets above, the following HTML page is used:

```
<HTML>

  <HEADER>
    <TITLE>JFreeChart : Basic Servlet Demo</TITLE>
  </HEADER>

  <BODY>
    <H2>JFreeChart: Basic Servlet Demo</H2>
    <P>
    There are two sample servlets available:
    <ul>
      <li>a very basic servlet to generate a <a
        href="servlet/ServletDemo1">bar chart</a>;</li>
      <li>another servlet that allow you to select one of <a
        href="chart.html">three sample charts</a>.  The selected chart is
        displayed in an HTML page.</li>
    </ul>
  </BODY>

</HTML>
```

There are two hyperlinks in this page, the first references the first demo servlet (`ServletDemo1`) and the second references another HTML page:

```
<HTML>

  <HEADER>
    <TITLE>JFreeChart Servlet Demo 2</TITLE>
  </HEADER>

  <BODY>
    <H2>JFreeChart Servlet Demo</H2>
    <P>
    Please choose a chart type:
    <FORM ACTION="servlet/ServletDemo2" METHOD=POST>
        <INPUT TYPE="radio" NAME="chart" VALUE="pie" CHECKED> Pie Chart
        <INPUT TYPE="radio" NAME="chart" VALUE="bar"> Bar Chart
        <INPUT TYPE="radio" NAME="chart" VALUE="time"> Time Series Chart
        <P>
        <INPUT TYPE="submit" VALUE="Generate Chart">
    </FORM>
  </BODY>

</HTML>
```

This second HTML page contains a `<FORM>` element used to specify a parameter for the second servlet (`ServletDemo2`). When this servlet runs, it returns its own HTML that is almost identical to the above but also includes an `<IMG>` element with a reference to the `ServletDemo2ChartGenerator` servlet.

available in the latter class.

15.5 Deploying Servlets

After compiling the demo servlets, they need to be deployed to a servlet engine, along with the supporting files, so that they can be accessed by clients. Fortunately, this is relatively straightforward.

The first requirement is a `web.xml` file to describe the web application being deployed:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
  <servlet>
    <servlet-name>
      ServletDemo1
    </servlet-name>
    <servlet-class>
      com.jrefinery.chart.premium.demo.ServletDemo1
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2
    </servlet-name>
    <servlet-class>
      com.jrefinery.chart.premium.demo.ServletDemo2
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2ChartGenerator
    </servlet-name>
    <servlet-class>
      com.jrefinery.chart.premium.demo.ServletDemo2ChartGenerator
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      ServletDemo1
    </servlet-name>
    <url-pattern>
      /servlet/ServletDemo1
    </url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>
      ServletDemo2
    </servlet-name>
    <url-pattern>
      /servlet/ServletDemo2
    </url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>
      ServletDemo2ChartGenerator
    </servlet-name>
    <url-pattern>
      /servlet/ServletDemo2ChartGenerator
    </url-pattern>
  </servlet-mapping>

</web-app>
```

This file lists the servlets by name, and specifies the class file that implements the servlet. The actual class files will be placed in a directory where the servlet engine will know to find them (the `classes` sub-directory within a directory specific to the application).

The final step is copying all the files to the appropriate directory for the servlet engine. In testing with Tomcat, I created a `jfreechart` directory within Tomcat's `webapps` directory. The `index.html` and `chart.html` files are copied to this directory.

```
webapps/jfreechart/index.html
webapps/jfreechart/chart.html
```

Next, a subdirectory WEB-INF is created within the `jfreechart` directory, and the `web.xml` file is copied to here.

```
webapps/jfreechart/WEB-INF/web.xml
```

A `classes` subdirectory is created within `WEB-INF` to hold the `.class` files for the three demo servlets. These need to be saved in a directory hierarchy matching the package hierarchy:

```
webapps/jfreechart/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo1.class
webapps/jfreechart/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo2.class
webapps/jfreechart/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemoChartGenerator.class
```

Finally, the servlets make use of classes in the JFreeChart and JCommon class libraries. The jar files for these libraries need to be added to a `lib` directory within `WEB-INF`. You will need:

```
webapps/jfreechart/WEB-INF/lib/jcommon-0.7.2.jar
webapps/jfreechart/WEB-INF/lib/jfreechart-0.9.6.jar
```

Now restart your servlet engine, and point your browser to:

```
http://localhost:8080/jfreechart/index.html
```

If all the files have been put in the correct places, you should see the running servlet demonstration.

16 Packages

16.1 Overview

The following sections contain reference information for the classes, arranged by package, that make up the JFreeChart class library.

Package:	Description:
o.j.chart	The main chart classes.
o.j.chart.annotations	A simple framework for annotating charts.
o.j.chart.axis	Axis classes and related interfaces.
o.j.chart.entity	Classes representing chart entities.
o.j.chart.event	The event classes.
o.j.chart.needle	Needle classes for the compass plot.
o.j.chart.plot	Plot classes and interfaces.
o.j.chart.renderer	Plug-in renderers for use with the CategoryPlot and XYPlot classes.
o.j.chart.servlet	Servlet utility classes.
o.j.chart.tooltips	The tooltip classes.
o.j.chart.urls	Interfaces and classes for generating URLs in image maps.
o.j.chart.ui	User interface classes.
o.j.data	Dataset interfaces and classes.
o.j.data.time	Time-based dataset interfaces and classes.

Additional information can be found in the Javadoc HTML files.

17 Package: org.jfree.chart

17.1 Overview

This package contains the major classes and interfaces in the JFreeChart class library.

17.2 AbstractTitle

17.2.1 Overview

The base class for all chart titles. Several concrete sub-classes have been implemented, including: `TextTitle`, `DateTitle` and `ImageTitle`.

The `JFreeChart` class maintains one chart title (an instance of `TextTitle`, `null` permitted) plus a list of subtitles (which can be any subclass of `AbstractTitle`).

When a chart is drawn, the title and/or subtitles will "grab" a rectangular section of the chart area in which to draw themselves. This reduces the amount of space for plotting data—so although there is no limit to the number of subtitles you can add to a chart, for practical reasons you need to keep the number reasonably low.

17.2.2 Constructors

This is an abstract class, so you won't instantiate it directly. However, the following constructor is available for subclasses to use:

> `protected AbstractTitle(int position, int horizontalAlignment, int verticalAlignment, Spacer spacer);`
> Creates a new `AbstractTitle`. The position should be one of: `TOP`, `BOTTOM`, `LEFT` or `RIGHT` (constants defined by this class).

17.2.3 Methods

You can set the "position" for a title:

> `public void setPosition(int position);`
> Sets the position for the title. Use one of the constants `TOP`, `BOTTOM`, `LEFT` or `RIGHT` defined by this class.

Within the rectangular area allocated for the title, you can specify the horizontal alignment:

> `public void setHorizontalAlignment(int alignment);`
> Sets the horizontal alignment for the title. Use one of the constants `LEFT`, `RIGHT` or `CENTER` defined by this class.

Similarly, you can specify the vertical alignment:

> `public void setVerticalAlignment(int alignment);`
> Sets the vertical alignment for the title. Use one of the constants `TOP`, `BOTTOM` or `MIDDLE` defined by this class.

17.2.4 Notes

The original version of this class was written by David Berry. I've since made a few changes to the original version, but the idea for allowing a chart to have multiple titles came from David.

The `JFreeChart` class implements the `TitleChangeListener` interface, and receives notification whenever a chart title is changed (this, in turn, triggers a `ChartChangeEvent` which usually results in the chart being redrawn).

This class implements `Cloneable`, which is useful when editing title properties because you can edit a copy of the original, and then either apply the changes or cancel the changes.

See Also

`DateTitle`, `ImageTitle`, `TextTitle`.

17.3 ChartColor

17.3.1 Overview

Not yet documented.

17.4 ChartFactory

17.4.1 Overview

This class provides a range of static methods for constructing charts. These methods make it easier to create charts with default properties.

17.4.2 Methods

```
public static JFreeChart createPieChart(String title, PieDataset data,
boolean legend);
```
Creates a *pie chart* for the given `PieDataset`.

```
public static JFreeChart createVerticalBarChart(String title,
String categoryAxisLabel, String valueAxisLabel,
CategoryDataset data, boolean legend);
```
Creates a *vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createVerticalBarChart3D(String title, String
categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean
legend);
```
Creates a *vertical bar chart with 3D effect* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart(String title, String
categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean
legend);
```
Creates a *stacked vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart3D(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```
Creates a *stacked vertical bar chart with 3D effect* for the given `CategoryDataset`.

`public static JFreeChart createHorizontalBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *horizontal bar chart* for the given `CategoryDataset`.

`public static JFreeChart createStackedHorizontalBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *stacked horizontal bar chart* for the given `CategoryDataset`.

`public static JFreeChart createLineChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *line chart* for the given `CategoryDataset`.

`public static JFreeChart createXYChart(String title, String xAxisLabel, String yAxisLabel, XYDataset data, boolean legend)`
Creates an *XY plot* for the given `XYDataset`.

`public static JFreeChart createScatterPlot(String title, String xAxisLabel, String yAxisLabel, XYDataset data, boolean legend)`
Creates a *scatter plot* for the given `XYDataset`.

`public static JFreeChart createTimeSeriesChart(String title, String timeAxisLabel, String valueAxisLabel, XYDataset data, boolean legend)`
Creates a *time series chart* for the given `XYDataset`.

`public static JFreeChart createVerticalXYBarChart(String title, String xAxisLabel, String yAxisLabel, IntervalXYDataset data, boolean legend)`
Creates a *vertical XY bar chart* for the given `IntervalXYDataset`.

`public static JFreeChart createHighLowChart(String title, String timeAxisLabel, String valueAxisLabel, HighLowDataset data, boolean legend)`
Creates a *high-low-open-close chart* for the given `HighLowDataset`.

`public static JFreeChart createCandlestickChart(String title, String timeAxisLabel, String valueAxisLabel, HighLowDataset data, boolean legend)`
Creates a *candlestick chart* for the given `HighLowDataset`.

17.4.3 Notes

These methods are provided for convenience only. You are not required to use them.

See Also
`JFreeChart`.

17.5 ChartFrame

17.5.1 Overview

A frame containing chart within a `ChartPanel`.

17.5.2 Constructors

There are two constructors:

```
public ChartFrame(String title, JFreeChart chart);
```
Creates a new `ChartFrame` containing the specified chart.

The second constructor gives you the opportunity to request that the chart is contained within a `JScrollPane`:

```
public ChartFrame(String title, JFreeChart chart, boolean scrollPane);
```
Creates a new `ChartFrame` containing the specified chart.

17.5.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`ChartPanel`.

17.6 ChartMouseEvent

17.6.1 Overview

An event generated by the `ChartPanel` class for mouse clicks and mouse movements over a chart.

17.6.2 Notes

To receive notification of these events, an object first needs to implement the `ChartMouseListener` interface and then register itself with a `ChartPanel` object.

See Also

`ChartPanel`, `ChartMouseListener`.

17.7 ChartMouseListener

17.7.1 Overview

An interface that defines the callback method for a *chart mouse listener*.

17.7.2 Methods

This receives notification of mouse click events:

```
public void chartMouseClicked(ChartMouseEvent event);
```
A callback method for receiving notification of a mouse click on a chart.

This method receives notification of mouse movement events:

```
public void chartMouseMoved(ChartMouseEvent event);
```
A callback method for receiving notification of a mouse movement event on a chart.

17.7.3 Notes

Any class that implements this interface can register with a `ChartPanel` object to receive notification of *chart mouse events*.

See Also

`ChartPanel`, `ChartMouseEvent`.

17.8 ChartPanel

17.8.1 Overview

A panel that provides a convenient means to display a `JFreeChart` instance in a Swing-based user-interface (extends `javax.swing.JPanel`).

The panel can be set up to include a popup menu providing access to:

- chart properties – the property editors are incomplete, but allow you to customise many chart properties;
- printing – print a chart via the standard Java printing facilities;
- saving – write the chart to a PNG format file;
- zooming – zoom in or out by adjusting the axis ranges;

In addition, the panel can:

- provide offscreen buffering to improve performance when redrawing overlapping frames;
- display tool tips;

All of these features are used in the demonstration applications included with the JFreeChart distribution.

17.8.2 Constructors

The standard constructor accepts a `JFreeChart` as the only parameter, and creates a panel that displays the chart:

```
public ChartPanel(JFreeChart chart);
```
Creates a new `ChartPanel` for drawing the specified chart.

By default, the panel is automatically updated whenever the chart changes (for example, if you modify the range for an axis, the chart will be redrawn automatically).

17.8.3 Methods

You can get access to the chart that is displayed in the panel:

`public JFreeChart getChart();`
Returns the chart that is displayed in the panel.

You can change the chart that is displayed in the panel:

`public void setChart(JFreeChart chart);`
Sets the chart that is displayed in the panel. The panel registers with the chart as a change listener, so that it can repaint the chart whenever it changes.

The panel includes support for displaying tool tips (assuming that tool tips have been generated by the plot or renderer). To disable (or re-enable) the display of tool tips, use the following method:

`public void setDisplayToolTips(boolean flag);`
Switches the display of tool tips on or off for this panel.

As the space available for drawing a chart gets smaller and smaller, it becomes more and more difficult to layout the components of the chart without overlaps. One solution to this is to draw a distinction between the chart *drawing size* and the chart *display size*. If the space on the panel is less than the minimum drawing size, then the chart is drawn in a buffer at the minimum size, then scaled (down) into the available space on the panel (the display size). Use the following method to specify the minimum drawing width:

`public void setMinimumDrawWidth(double width);`
Sets the minimum width for drawing the chart. A scaling transformation is used to fit the chart into spaces smaller than this, if required.

...and this method to set the minimum drawing height:

`public void setMinimumDrawHeight(double height);`
Sets the minimum height for drawing the chart. A scaling transformation is used to fit the chart into spaces smaller than this, if required.

17.8.4 Notes

The size of the `ChartPanel` is determined by the layout manager used to arrange components in your user interface. In some cases, the layout manager will respect the *preferred size* of the panel, which you can set like this:

```
myChartPanel.setPreferredSize(new Dimension(500, 270));
```

This class implements the `Printable` interface, to provide a simple mechanism for printing a chart. An option in the panel's popup menu calls the `createPrintJob()` method. The print job ends up calling the `print(...)` method to draw the chart on a single piece of paper.

If you need greater control over the printing process—for example, you want to display several charts on one page—you can write your own implementation of

the `Printable` interface (in any class that has access to the chart(s) you want to print). The implementation incorporated with the `ChartPanel` class is a basic example, provided for convenience only.

The chart panel provides a "mouse zooming" feature. A demonstration of this is provided in the `MouseZoomDemo` application.

See Also

`JFreeChart`.

17.9 ChartPanelConstants

17.9.1 Overview

An interface that defines constants used by the `ChartPanel` class.

17.10 ChartRenderingInfo

17.10.1 Overview

This class can be used to collect information about a chart as it is rendered, particularly information concerning the dimensions of various sub-components of the chart.

In the current implementation, four pieces of information are recorded for most chart types:

- the chart area;
- the plot area (including the axes);
- the data area ("inside" the axes);
- the dimensions are other information (including tool tips) for the entities within a chart;

You have some control over the information that is generated. For instance, tool tips will not be generated unless you set up a generator in the renderer.

17.10.2 Constructors

The default constructor:

> `public ChartRenderingInfo();`
> Creates a `ChartRenderingInfo` object. Entity information will be collected using an instance of `StandardEntityCollection`.

An alternative constructor allows you to supply a specific entity collection:

> `public ChartRenderingInfo(EntityCollection entities);`
> Creates a `ChartRenderingInfo` object.

17.10.3 Notes

The `ChartPanel` class automatically collects entity information using this class, because it needs it to generate tool tips.

See Also
EntityCollection.

17.11 ChartUtilities

17.11.1 Overview

This class contains some utility methods for converting charts to various image formats, including PNG, JPEG and HTML image maps. All of the methods in this class are `static`.

17.11.2 Methods

To save a chart in the *Portable Network Graphics* (PNG) format:

> `public static void saveChartAsPNG(File file, JFreeChart chart, int width, int height);`
> Saves a chart to a PNG format image file.

If you need to know more information about the structure of the chart within the generated image, you will need to pass in a `ChartRenderingInfo` object:

> `public static void saveChartAsPNG(File file, JFreeChart chart, int width, int height, ChartRenderingInfo info);`
> Saves a chart to a PNG format image file. If an `info` object is supplied, it will be populated with information about the structure of the chart.

Similar methods are available for writing a chart to an `OutputStream` in PNG format (two `writeChartAsPNG(...)` methods).

To save a chart in the *Joint Photographic Experts Group* (JPEG) format:

> `public static void saveChartAsJPEG(File file, JFreeChart chart, int width, int height);`
> Saves a chart to a JPEG format image file.

As with the PNG methods, if you need to know more information about the structure of the chart within the generated image, you will need to pass in a `ChartRenderingInfo` object:

> `public static void saveChartAsJPEG(File file, JFreeChart chart, int width, int height, ChartRenderingInfo info);`
> Saves a chart to a JPEG format image file. If an `info` object is supplied, it will be populated with information about the structure of the chart.

You can generate a simple HTML image map:

> `public static void writeImageMap(PrintWriter writer, String name, String hrefPrefix, ChartRenderingInfo info);`
> Writes a `<MAP>` element containing the region definitions for a chart that has been converted to an image. The `info` object should be the structure returned from the method call that wrote the chart to an image file.

There are two demonstration applications in the JFreeChart download that illustrate how this works: `ImageMapDemo1` and `ImageMapDemo2`.

17.11.3 Notes

PNG tends to be a better format for charts than JPEG since the compression is "lossless" for PNG.

See Also

JFreeChart, ChartRenderingInfo.

17.12 ClipPath

17.12.1 Overview

Not yet documented.

17.13 CrosshairInfo

17.13.1 Overview

This class maintains information about the crosshairs on a plot, as the plot is being rendered.

17.13.2 Constructors

The default constructor:

```
public CrosshairInfo();
```
Creates a `CrosshairInfo` object.

17.13.3 Methods

The following method is called as a plot is being rendered:

```
public void updateCrosshairPoint(double candidateX, double candidateY);
```
Creates a `CrosshairInfo` object.

17.14 DateTitle

17.14.1 Overview

A chart title that displays the current date. Since charts can have multiple titles, this class enables the current date to be added in various positions relative to the chart (often at the bottom).

17.14.2 Notes

The original version of this class was written by David Berry (`dberry@dallas.net`).

See Also

AbstractTitle.

17.15 DefaultShapeFactory

17.15.1 Overview

A *shape factory* implementation provided to match the behaviour of older versions of JFreeChart. You should use `SeriesShapeFactory` instead.

17.16 DrawableLegendItem

17.16.1 Overview

Not yet documented.

17.17 Effect3D

17.17.1 Overview

Not yet documented.

17.18 ImageTitle

17.18.1 Overview

A chart title that displays an image. Extends `AbstractTitle`.

17.18.2 Constructors

To create an image title:

```
public ImageTitle(Image image);
```
Creates an image title. By default, the title is positioned at the top of the chart, and the image is centered horizontally within the available space.

17.18.3 Notes

This class was written and contributed by David Berry.

See Also

`AbstractTitle`.

17.19 IntervalMarker

17.19.1 Overview

Not yet documented.

17.20 JFreeChart

17.20.1 Overview

The `JFreeChart` class coordinates the entire process of drawing charts. One method:

```
public void draw(Graphics2D g2, Rectangle2D area);
```

...instructs the JFreeChart object to draw a chart onto a specific area on some *graphics device*.

Java supports several graphics devices—including the screen, the printer, and buffered images—via different implementations of the abstract class java.awt.Graphics2D. Thanks to this abstraction, JFreeChart can generate charts on any of these target devices, as well as others implemented by third parties (for example, the SVG Generator implemented by the Batik Project).

In broad terms, JFreeChart sets up a context for drawing a Plot. The plot obtains data from a Dataset, and may delegate the drawing of individual data items to a CategoryItemRenderer or an XYItemRenderer, depending on the plot type.

The JFreeChart class can work with many different Plot subclasses. Depending on the type of plot, a specific dataset will be required. The following table summarises the combinations that are currently available:

Dataset:	Compatible Plot Types:
MeterDataset	CompassPlot, MeterPlot and ThermometerPlot.
PieDataset	PiePlot.
CategoryDataset	CategoryPlot subclasses with various renderers.
XYDataset	XYPlot with various renderers.
IntervalXYDataset	XYPlot with a VerticalXYBarRenderer.
HighLowDataset	XYPlot with a HighLowRenderer.
HighLowDataset	XYPlot with a CandleStickRenderer.

17.20.2 Constructors

All constructors require you to supply a Plot instance (the Plot maintains a reference to the dataset used for the chart).

The simplest constructor is:

```
public JFreeChart(Plot plot);
```
Creates a new JFreeChart instance. The chart will have no title, and no legend.

For greater control, a more complete constructor is available:

```
public JFreeChart(Plot plot, String title, Font titleFont, boolean createLegend);
```
Creates a new JFreeChart instance. This constructor allows you to specify a single title (you can add additional titles, later, if necessary).

The ChartFactory class provides some utility methods that can make the process of constructing charts simpler.

17.20.3 Attributes

The attributes maintained by the JFreeChart class are listed in Table 1.

17.20.4 Methods

The most important method for a chart is the draw(...) method:

Attribute:	Description:
title	The chart title (an instance of `TextTitle`).
sub-titles	A list of subtitles.
legend	The chart legend.
plot	The plot.
antialias	A flag that indicates whether or not the chart should be drawn with anti-aliasing.
background-paint	The background paint for the chart.
background-image	An optional background image for the chart.
background-image-alignment	The alignment of the background image (if there is one).
background-image-alpha	The alpha transparency for the background image.

Table 1: Attributes for the `JFreeChart` class

```
public void draw(Graphics2D g2, Rectangle2D chartArea);
```
Draws the chart on the `Graphics2D` device, within the specified area.

The chart does not retain any information about the location or dimensions of the items it draws. Callers that require such information should use the alternative method:

```
public void draw(Graphics2D g2, Rectangle2D chartArea, ChartRenderingInfo
info);
```
Draws the chart on the `Graphics2D` device, within the specified area. If `info` is not `null`, it will be populated with information about the items drawn within the chart (to be returned to the caller).

Charts can have zero, one or many titles. To add a title to the chart:

```
public void addTitle(AbstractTitle title);
```
Adds a title to the chart.

The legend shows the names of the series (or sometimes categories) in a chart, next to a small color indicator. To set the legend for a chart:

```
public void setLegend(Legend legend);
```
Sets the legend for a chart.

You can control whether or not the chart is drawn with anti-aliasing (switching anti-aliasing *on* can improve the on-screen appearance of charts):

```
public void setAntiAlias(boolean flag);
```
Sets a flag controlling whether or not anti-aliasing is used when drawing the chart.

To receive notification of any change to a chart, a listener object should register via this method:

```
public void addChangeListener(ChartChangeListener listener);
```
Register to receive chart change events.

To stop receiving change notifications, a listener object should deregister via this method:

```
public void removeChangeListener(ChartChangeListener listener);
```
Deregister to stop receiving chart change events.

17.20.5 Notes

The `ChartFactory` class provides a large number of methods for creating "ready-made" charts.

The Java2D API is used throughout JFreeChart, so JFreeChart does not work with JDK1.1 (a common question from applet developers, although hopefully less of an issue as browser support for Java 2 improves).

17.21 JFreeChartConstants

17.21.1 Overview

A collection of constants used by the JFreeChart class.

See Also

`JFreeChart`.

17.22 Legend

17.22.1 Overview

The base class for a *chart legend* (displays the series names and colors used in a chart). The legend can appear at the top, bottom, left or right of a chart.

`StandardLegend` is the only subclass available.

17.22.2 Usage

If you create charts using the `ChartFactory` class, a legend will often be created for you. You can access the legend using the `getLegend()` method in the `JFreeChart` class.

To change the position of the legend relative to the chart to one of the positions `NORTH`, `SOUTH`, `EAST` or `WEST`, use the following code:

```
Legend legend = myChart.getLegend();
legend.setAnchor(Legend.WEST);
```

If you don't want a legend to appear on your chart, you can set it to `null`:

```
myChart.setLegend(null);
```

17.22.3 Constructor

This is an abstract class, so the constructor is `protected`.

17.22.4 Notes

This class implements a listener mechanism which can be used by subclasses.

See Also
StandardLegend.

17.23 LegendItem

17.23.1 Overview

An item within a legend.

See Also
Legend.

17.24 LegendItemCollection

17.24.1 Overview

A collection of legend items.

See Also
Legend.

17.25 LegendItemLayout

17.25.1 Overview

An interface for laying out a collection of legend items.

17.25.2 Notes

This code is incomplete.

See Also
Legend.

17.26 LegendTitle

17.26.1 Overview

Not yet documented.

17.27 Marker

17.27.1 Overview

Represents a constant value to be "marked" on a plot. Most plots will draw a line across the plot to indicate the marker.

See Also
CategoryPlot, XYPlot.

17.28 MeterLegend

17.28.1 Overview

To be documented.

17.29 PaintTable

17.29.1 Overview

An interface that can be used to look up `Paint` objects in a two-dimensional table.

17.30 SeriesShapeFactory

17.30.1 Overview

An implementation of the `ShapeFactory` interface that generates shapes for use on charts.

17.31 ShapeFactory

17.31.1 Overview

An interface for generating shapes for a chart. To be documented.

17.32 Spacer

17.32.1 Overview

This class is used to specify left, right, top and bottom margins relative to an arbitrary rectangle. The space can be specified in absolute terms (points, or 1/72 inch) or relative terms (a percentage of the height or width of the rectangle).

17.32.2 Constructor

To create a new `Spacer`:

> `public Spacer(int type, double left, double top, double right, double left);`
> Creates a new spacer. The `type` can be `ABSOLUTE` or `RELATIVE`. The remaining arguments are interpreted as points (1/72 inch) for absolute spacing, or percentages for relative spacing.

17.32.3 Methods

To get the amount of spacing for the left side:

> `public double getLeftSpace(double width);`
> Returns the amount of spacing for the left side.

To get the amount of spacing for the right side:

> `public double getRightSpace(double width);`
> Returns the amount of spacing for the right side.

In both of the above methods, the `width` argument refers to the width of a rectangle that the space calculation is relative to. It is ignored if the space is specified in absolute terms.

To get the amount of spacing for the top side:

`public double getTopSpace(double height);`
Returns the amount of spacing for the top side.

To get the amount of spacing for the bottom side:

`public double getBottomSpace(double height);`
Returns the amount of spacing for the top side.

In both of the above methods, the `height` argument refers to the height of a rectangle that the space calculation is relative to. It is ignored if the space is specified in absolute terms.

A given rectangle can be "shrunk" by a spacer object:

`public void trim(Rectangle2D area);`
Reduces the dimensions of the specified `area`, according to the space settings.

17.32.4 Notes

Throughout JFreeChart, the `Insets` class has been used to specify (absolute) padding information. This class is intended to replace the use of `Insets` to allow both absolute and relative settings.

17.33 StandardLegend

17.33.1 Overview

A chart legend displays the names of the series in a chart.

17.33.2 Notes

It is planned that this class should be replaced by a `LegendTitle` class, so that the legend can be treated (for layout purposes) as if it were a chart title.

17.34 StandardLegendItemLayout

17.34.1 Overview

Not yet documented.

17.35 TextTitle

17.35.1 Overview

A text-based chart title (extends `AbstractTitle`). The title can appear at the `TOP` or `BOTTOM` of a chart, but (for now) not at the `LEFT` or `RIGHT`.

17.35.2 Constructors

To create a text title for a chart:

`public TextTitle(String text);`
Creates a chart title using the specified text. By default, the title will be positioned at the top of the chart, centered horizontally. The font defaults to `SansSerif`, 12pt bold and the color defaults to black.

There are other constructors that provide more control over the attributes of the `TextTitle`.

17.35.3 Methods

To set the title string:

`public void setText(String text);`
Sets the text for the title and notifies registered listeners that the title has changed.

To set the font for the title:

`public void setFont(Font font);`
Sets the font for the title and notifies registered listeners that the title has changed.

To set the color of the title:

`public void setPaint(Paint paint);`
Sets the paint used to display the title text, and notifies registered listeners that the title has changed.

The following method is called by the `JFreeChart` class to actually draw the chart title:

`public void draw(Graphics2D g2, Rectangle2D area);`
Draws the title onto a graphics device, to occupy the specified area.

There are additional methods inherited from the `AbstractTitle` class.

17.35.4 Notes

This class does not support multi-line text (yet). You should use multiple subtitles if you need to display more text than you can fit onto one line.

The title string can contain any characters from the Unicode character set. However, you need to ensure that the `Font` that you use to display the title actually supports the characters you want to display. Most fonts do not support the full range of Unicode characters, but this website has some information about fonts that you might be able to use:

`http://www.ccss.de/slovo/unifonts.htm`

The original version of this class was written by David Berry.

See Also

`AbstractTitle`.

18 Package: org.jfree.chart.annotations

18.1 Overview

A basic set of classes for adding annotations to charts. In the current release, you can add a text annotation to an `XYPlot`. This is more or less a "proof of concept", and it is planned that future releases will extend this framework.

18.2 Annotation

18.2.1 Overview

The base interface for plot annotations. Extensions of this interface include:

- `XYAnnotation` – an annotation that can be added to an `XYPlot`;

18.2.2 Notes

This interface defines no methods. It serves at the root interface for a hierarchy of related interfaces.

18.3 CategoryAnnotation

18.3.1 Overview

Not yet documented.

18.4 CategoryTextAnnotation

18.4.1 Overview

Not yet documented.

18.5 TextAnnotation

18.5.1 Overview

The base class for a *text annotation*. Subclasses will add location information to the content represented by this class.

18.5.2 Constructor

The constructor for this class is `protected` since you won't create an instance of this class directly (use a subclass):

```
protected TextAnnotation(String text, Font font, Paint paint);
```
Creates a new text annotation with the specified attributes.

18.5.3 Methods

There are methods for accessing the `text`, `font` and `paint` attributes. Setter methods are not required by the interface, although classes that implement the interface may provide them.

18.6 XYAnnotation

18.6.1 Overview

The interface that must be supported by annotations that are to be added to an `XYPlot` (extends the `Annotation` interface).

This interface is implemented by:

- `XYTextAnnotation`;

You can, of course, provide your own implementations of the interface.

18.6.2 Methods

This class defines one method for drawing the annotation:

```
public void draw(Graphics2D g2, Rectangle2D dataArea,
ValueAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation. The `dataArea` is the space defined by (within) the two axes. If the annotation defines its location in terms of data values, the axes can be used to convert these values to Java2D coordinates.

18.7 XYLineAnnotation

18.7.1 Overview

Not yet documented.

18.8 XYTextAnnotation

18.8.1 Overview

A text annotation that can be added to an `XYPlot` (extends the `TextAnnotation` class).

18.8.2 Usage

To add an annotation to an `XYPlot`:

```
XYPlot plot = myChart.getXYPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

19 Package: org.jfree.chart.axis

19.1 Overview

The `org.jfree.chart.axis` package contains all the axis classes plus a few assorted support classes and interfaces.

19.2 Axis

19.2.1 Overview

An abstract base class representing an axis. Some subclasses of `Plot`, including `CategoryPlot` and `XYPlot`, will use axes to display data.

Figure 3 illustrates the axis class hierarchy.

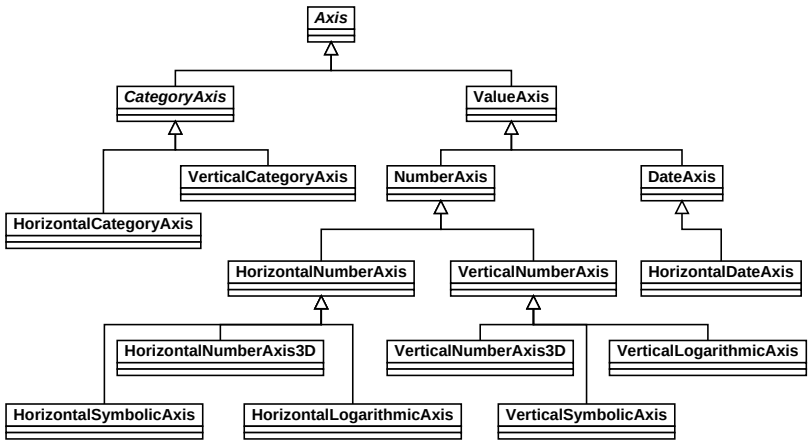


Figure 3: Axis classes

19.2.2 Constructors

The constructors for this class are `protected`, you cannot create an instance of this class directly—you must use a subclass.

19.2.3 Attributes

The attributes maintained by the `Axis` class are listed in Table 2. There are methods to read and update most of these attributes. In most cases, updating an axis attribute will result in an `AxisChangeEvent` being sent to all (or any) registered listeners.

The default values used to initialise the axis attributes are listed in Table 3.

19.2.4 Usage

To change the attributes of an axis, you must first obtain a reference to the axis. Usually, you will obtain the reference from the plot that uses the axis. For example:

Attribute:	Description:
plot	The plot to which the axis belongs.
visible	A flag that controls whether or not the axis is visible.
label	The axis label.
label-font	The font for the axis label.
label-paint	The foreground color for the axis label.
label-insets	The space to leave around the outside of the axis label.
tick-labels-visible	A flag controlling the visibility of tick labels.
tick-label-font	The font for the tick labels.
tick-label-paint	The color for the tick labels.
tick-label-insets	The space to leave around the outside of the tick labels.
tick-marks-visible	A flag controlling the visibility of tick marks.
tick-mark-stroke	The stroke used to draw the tick marks.
tick-mark-paint	The paint used to draw the tick marks.
tick-mark-inside-length	The amount by which the tick marks extend into the plot area.
tick-mark-outside-length	The amount by which the tick marks extend outside the plot area.

Table 2: Attributes for the `Axis` class

Name:	Value:
DEFAULT_AXIS_LABEL_FONT	`new Font("SansSerif", Font.PLAIN, 14);`
DEFAULT_AXIS_LABEL_PAINT	`Color.black;`
DEFAULT_AXIS_LABEL_INSETS	`new Insets(2, 2, 2, 2);`
DEFAULT_TICK_LABEL_FONT	`new Font("SansSerif", Font.PLAIN, 10);`
DEFAULT_TICK_LABEL_PAINT	`Color.black;`
DEFAULT_TICK_LABEL_INSETS	`new Insets(2, 1, 2, 1);`
DEFAULT_TICK_STROKE	`new BasicStroke(1);`

Table 3: `Axis` class default attribute values

```
CategoryPlot plot = myChart.getCategoryPlot();
CategoryAxis axis = plot.getDomainAxis();
// change axis attributes here...
```

Notice that the `getDomainAxis()` method returns a particular subclass of `Axis` (`CategoryAxis` in this case). That's okay, because the subclass inherits all the attributes defined by `Axis` anyway.

19.2.5 Change Notification

This class implements a *change notification mechanism* that is used to notify other objects whenever an axis is changed in some way. This is part of a JFreeChart-wide mechanism that makes it possible to receive notifications whenever a component of a chart is changed. Most often, such notifications result in the chart being redrawn.

The following methods are used:

```
public void addChangeListener(AxisChangeListener listener);
```
Registers an object to receive notification whenever the axis changes.

`public void removeChangeListener(AxisChangeListener listener);`
Deregisters an object, so that it no longer receives notification when the axis changes.

`public void notifyListeners(AxisChangeEvent event);`
Notifies all registered listeners that a change has been made to the axis.

See Also
`AxisConstants`, `AxisChangeEvent`, `AxisChangeListener`, `AxisNotCompatibleException`.

19.3 AxisConstants

19.3.1 Overview

An interface that defines the constants used by the `Axis` class.

19.3.2 Notes

The `Plot` class also implements this interface, so that it has convenient access to the constants for internal use.

See Also
`Axis`.

19.4 AxisNotCompatibleException

19.4.1 Overview

An exception that indicates that an attempt has been made to assign an axis to a `Plot` where the axis is not compatible with the plot type (for example, a `VerticalCategoryAxis` will not work with an `XYPlot`).

19.4.2 Constructors

To create a new exception:

`public AxisNotCompatibleException(String message);`
Creates a new exception.

19.4.3 Notes

This exception is a subclass of `RuntimeException`.

See Also
`PlotNotCompatibleException`.

19.5 CategoryAxis

19.5.1 Overview

An abstract base class for axes that display categories—extends `Axis`. The domain axis for a `CategoryPlot` must be an instance of this class. There are two known subclasses, `HorizontalCategoryAxis` and `VerticalCategoryAxis`.

19.5.2 Constructor

The constructor for this class is `protected`, you cannot instantiate this class directly—you must use a subclass.

19.5.3 Attributes

The attributes maintained by the `CategoryAxis` class are listed in Table 4. These attributes are in addition to those inherited from the `Axis` class (see section 19.2.3 for details).

Attribute:	Description:
lower-margin	The margin that appears before the first category, expressed as a percentage of the overall axis length.
upper-margin	The margin that appears after the last category, expressed as a percentage of the overall axis length.
category-margin	The margin between categories, expressed as a percentage of the overall axis length (to be distributed between N-1 gaps, where N is the number of categories).

Table 4: Attributes for the `CategoryAxis` class

The following default values are used:

Default:	Value:
`DEFAULT_AXIS_MARGIN`	`0.05` (5 percent).
`DEFAULT_CATEGORY_MARGIN`	`0.20` (20 percent).

19.5.4 Notes

Tick marks are not supported by this class (in version 0.9.5).

See Also

`HorizontalCategoryAxis`, `VerticalCategoryAxis`.

19.6 ColorBarAxis

19.6.1 Overview

Not yet documented.

19.7 DateAxis

19.7.1 Overview

The base class for axes that display date/time values—extends `ValueAxis`. This class is designed to be flexible about the range of dates/times that it can display—anything from a few milliseconds to several centuries can be handled.

Subclasses include `HorizontalDateAxis` and `VerticalDateAxis`.

19.7.2 Constructors

The constructors for this class are `protected`. You cannot create an instance of this class directly, you must use a subclass.

19.7.3 Attributes

The following attributes are defined, in addition to those inherited from the `ValueAxis` class:

Attribute:	Description:
date-format-override	A date formatter that, if set, overrides the format of the tick labels displayed on the axis.
tick-unit	Controls the size and formatting of the tick labels on the axis (an instance of `DateTickUnit`).
minimum-date	The minimum date/time visible on the axis.
maximum-date	The maximum date/time visible on the axis.

Refer to section 19.26.3 for information about the attributes inherited by this class.

19.7.4 Usage

To change the attributes of the axis, you need to obtain a `DateAxis` reference—because of the way JFreeChart is designed, this usually involves a "cast":

```
XYPlot plot = myChart.getXYPlot();
ValueAxis domainAxis = plot.getDomainAxis();
if (domainAxis instanceof DateAxis) {
    DateAxis axis = (DateAxis) domainAxis;
    // customise axis here...
}
```

Given a `DateAxis` reference, you can change:

- the axis range, see section 19.7.5;
- the size and formatting of the tick labels, see section 19.7.6;
- other inherited attributes, see section 19.26.4.

19.7.5 The Axis Range

To set the axis range:[6]

```
// start and end are instances of java.util.Date
axis.setRange(start, end);
```

19.7.6 Tick Units

The tick units on the date axis are controlled by a similar "auto tick unit selection" mechanism to that used in the `NumberAxis` class. This mechanism relies on a collection of "standard" tick units (stored in an instance of `TickUnits`). The axis will try to select the smallest tick unit that doesn't cause the tick labels to overlap.

If you want to specify a fixed tick size and format, you can use code similar to this:

[6] Note that when you set the axis range in this way, the *auto-range* attribute is set to `false`. It is assumed that by setting a range manually, you do not want that subsequently overridden by the auto-range calculation.

```
// set the tick size to one week, with formatting...
DateFormat formatter = new SimpleDateFormat("d-MMM-yyyy");
DateTickUnit unit = new DateTickUnit(DateTickUnit.DAY, 7, formatter);
axis.setTickUnit(unit);
```

Note that setting a tick unit manually in this way disables the "auto" tick unit selection mechanism. You may find that the tick size you have requested results in overlapping labels.

If you just want to control the tick label format, one option is to specify an *override format*:

```
// specify an override format...
DateFormat formatter = new SimpleDateFormat("d-MMM");
axis.setDateFormatOverride(formatter);
```

This is a simple and effective approach in some situations, but has the limitation that the same format is applied to all tick sizes.

A final approach to controlling the formatting of tick labels is to create your own `TickUnits` collection. The collection can contain any number of `DateTickUnit` objects, and should be registered with the axis as follows:

```
// supply a new tick unit collection...
axis.setStandardTickUnits(myCollection);
```

19.8 DateTickUnit

19.8.1 Overview

A date tick unit for use by subclasses of `DateAxis` (extends the `TickUnit` class).

The unit size can be specified as a multiple of one of the following time units:

Time Unit:	Constant:
Year	`DateTickUnit.YEAR`
Month	`DateTickUnit.MONTH`
Day	`DateTickUnit.DAY`
Hour	`DateTickUnit.HOUR`
Minute	`DateTickUnit.MINUTE`
Second	`DateTickUnit.SECOND`
Millisecond	`DateTickUnit.MILLISECOND`

Note that these constants are not the same as those defined by Java's `Calendar` class.

19.8.2 Usage

There are two ways to make use of this class. The first is where you know the exact tick size that you want for your axis. In this case, you create a new date tick unit then call the `setTickUnit(...)` method in the `DateAxis` class. For example, to set the tick unit size on the axis to one week:

```
XYPlot plot = myChart.getXYPlot();
ValueAxis axis = plot.getDomainAxis();
axis.setTickUnit(new DateTickUnit(DateTickUnit.DAY, 7));
```

The second usage is to create a collection of tick units using the `TickUnits` class, and then allow the `DateAxis` to automatically select an appropriate unit. See the `setStandardTickUnits(...)` method for more details.

19.8.3 Constructors

To create a new date tick unit:

`public DateTickUnit(int unit, int count);`
Creates a new tick unit with a default date formatter for the current locale.

Alternatively, you can supply your own date formatter:

`public DateTickUnit(int unit, int count, DateFormat formatter);`
Creates a new date tick unit with the specified date formatter.

For both constructors, the `unit` argument should be defined using one of the constants listed in section 19.8.1. The `count` argument specifies the multiplier (often just `1`).

19.8.4 Methods

To get the units used to specify the tick size:

`public int getUnit();`
Returns a constant representing the units used to specify the tick size. The constants are listed in section 19.8.1.

To get the number of units:

`public int getCount();`
Returns the number of units.

To format a date using the tick unit's internal formatter:

`public String dateToString(Date date);`
Formats the date as a `String`.

The following method is used for simple date addition:

`public Date addToDate(Date base);`
Creates a new `Date` that is calculated by adding this `DateTickUnit` to the `base` date.

19.8.5 Notes

This class is immutable, a requirement for all subclasses of `TickUnit`.

See Also
`NumberTickUnit`.

19.9 HorizontalAxis

19.9.1 Overview

An interface that *must* be implemented by all horizontal axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

19.9.2 Methods

The interface defines two methods—the plot will call one of these two methods, depending on the implementation.

The first method calculates the height required to display the axis without any knowledge of the width required by the vertical axis/axes (so an element of guess-work is involved):

> `public double reserveHeight(Graphics2D g2, Plot plot,`
> `Rectangle2D drawArea, int location);`
> Estimates the height that the horizontal axis requires to draw itself. The `location` will be either `TOP` or `BOTTOM`, and may or may not affect the result.

The second method is similar to the first except that the width required for the vertical axis/axes has already been calculated:

> `public double reserveHeight(Graphics2D g2,`
> `Plot plot, Rectangle2D drawArea, int location,`
> `double reservedWidth, int verticalAxisLocation);`
> Calculates the area that the horizontal axis requires to draw itself. The width required to draw the vertical axis/axes is given.

19.9.3 Notes

For vertical axes, the `VerticalAxis` interface performs a similar role.

19.10 HorizontalCategoryAxis

19.10.1 Overview

A horizontal axis that displays categories—extends `CategoryAxis` and implements `HorizontalAxis`.

This axis is used with the `VerticalCategoryPlot` class.

19.10.2 Constructors

To create a new axis:

> `public HorizontalCategoryAxis(String label);`
> Creates a new axis with the supplied label (`null` permitted).

The axis will be initialised with default values, you can subsequently change these if required.

19.10.3 Attributes

The `HorizontalCategoryAxis` class maintains the following attributes, in addition to those it inherits from `CategoryAxis` (refer to section 19.5.3 for details):

Attribute:	Description:
vertical-category-labels	A flag that controls whether the axis label is rotated to a "vertical" orientation.
skip-category-labels-to-fit	A flag that controls whether to skip some category labels so that the labels do not overlap.

The following default values are used:

Default:	Value:
`DEFAULT_VERTICAL_CATEGORY_LABELS`	`false`

19.10.4 Notes

There is a bug in Sun's JDK 1.3 that causes vertical labels to be mis-aligned. This does not occur in IBM's JDK 1.3, and has been fixed in Sun's JDK 1.4.

See Also

`VerticalCategoryAxis`.

19.11 HorizontalCategoryAxis3D

19.11.1 Overview

Not yet documented.

19.12 HorizontalColorBarAxis

19.12.1 Overview

Not yet documented.

19.13 HorizontalDateAxis

19.13.1 Overview

An axis that displays *date/time* values—extends `DateAxis` and implements `HorizontalAxis`.

With its horizontal orientation, this axis can be used as the domain axis for an `XYPlot` or the range axis for a `HorizontalCategoryPlot`.

19.13.2 Constructors

There are several constructors available, the most commonly used is:

`public HorizontalDateAxis(String label);`
Creates a new axis with the specified label (`null` permitted).

Refer to the Javadoc HTML files for information about the other constructors.

19.13.3 Attributes

This class adds the following attributes to those it inherits from the `DateAxis` class:

Attribute:	Description:
vertical-tick-labels	A flag that controls whether or not the tick labels on the axis are displayed "vertically" (that is, rotated 90 degrees from horizontal).

Refer to section 19.7.3 for information about the attributes inherited by this class.

19.13.4 Usage

To change the attributes of the axis, you need to obtain a `HorizontalDateAxis` reference—because of the way JFreeChart is designed, this usually involves a "cast":

```
XYPlot plot = myChart.getXYPlot();
ValueAxis domainAxis = plot.getDomainAxis();
if (domainAxis instanceof HorizontalDateAxis) {
    HorizontalDateAxis axis = (HorizontalDateAxis) domainAxis;
    // customise axis here...
}
```

Given a `HorizontalDateAxis` reference, you can change:

- the orientation of the tick labels, see section 19.13.5;
- general setting (the range, tick size and formatting, etc.), see section 19.7.4.

19.13.5 Tick Label Orientation

To control the orientation of the tick labels on the axis:

```
axis.setVerticalTickLabels(true);
```

Code similar to this can be used for all horizontal axes.

19.13.6 Notes

Although the axis displays dates for tick labels, at the lowest level it is still working with `double` primitives obtained from the `Number` objects supplied by the plot's dataset. The values are interpreted as *the number of milliseconds since 1 January 1970* (that is, the same encoding used by `java.util.Date`).

See Also

`VerticalDateAxis`.

19.14 HorizontalLogarithmicAxis

19.14.1 Overview

A numerical axis that displays values using a logarithmic scale.

19.14.2 Notes

This class is similar to the VerticalLogarithmicAxis class.

19.15 HorizontalLogarithmicColorBarAxis

19.15.1 Overview

Not yet documented.

19.16 HorizontalMarkerAxisBand

19.16.1 Overview

A band that can be added to a HorizontalNumberAxis to highlight certain value ranges.

19.16.2 Usage

To use this class, first create a new band:

```
HorizontalMarkerAxisBand band = new HorizontalMarkerAxisBand(
    axis, 2.0, 2.0, 2.0, 2.0,
    new Font("SansSerif", Font.PLAIN, 9));
```

Next, add as many ranges as you require to be displayed on the axis:

```
IntervalMarker m1 = new IntervalMarker(0.0, 33.0,
                                       "Low", Color.gray,
                                       new BasicStroke(0.5f),
                                       Color.green, 0.75f);
band.addMarker(m1);

IntervalMarker m2 = new IntervalMarker(33.0, 66.0,
                                       "Medium", Color.gray,
                                       new BasicStroke(0.5f),
                                       Color.orange, 0.75f);
band.addMarker(m2);

IntervalMarker m3 = new IntervalMarker(66.0, 100.0,
                                       "High", Color.gray,
                                       new BasicStroke(0.5f),
                                       Color.red, 0.75f);
band.addMarker(m3);
```

19.17 HorizontalNumberAxis

19.17.1 Overview

An horizontal axis that displays numerical data—this class extends NumberAxis and implements HorizontalAxis.

19.17.2 Constructors

There is a single constructor:

```
public HorizontalNumberAxis(String label);
```
Creates a new axis with the specified label (`null` permitted).

19.17.3 Attributes

This class defines the following attributes, in addition to those it inherits from the `NumberAxis` class:

Attribute:	Description:
vertical-tick-labels	A flag that indicates whether or not the tick labels are rotated to vertical.
marker-band	An optional band that highlights ranges along the axis (see `HorizontalMarkerAxisBand`).

19.17.4 Methods

Some notes on the methods in `HorizontalNumberAxis`:

`public void autoAdjustRange();`
Obtains the minimum and maximum data values from the `Plot`, provided that it implements `HorizontalValueRange`, and adjusts the axis range accordingly. Note that the `autoRangeIncludesZero` flag is checked in this method.

`public void refreshTicks(...);`
A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the `autoTickUnits` flag, and automatically determines a suitable "standard" tick size if required.

19.17.5 Notes

Refer to the Javadoc HTML files and the source code for details.

See Also
`VerticalNumberAxis`.

19.18 HorizontalNumberAxis3D

19.18.1 Overview

A horizontal number axis that works with the horizontal 3D bar chart.

19.19 HorizontalSymbolicAxis

19.19.1 Overview

An axis that displays numerical data using symbols.

See Also
`HorizontalNumberAxis`.

19.20 NumberAxis

19.20.1 Overview

The base class for axes that display numerical data (this class extends `ValueAxis`).

Commonly used subclasses include:

- `HorizontalNumberAxis`;
- `HorizontalLogarithmicAxis`;
- `VerticalNumberAxis`;
- `VerticalLogarithmicAxis`.

You can create your own subclasses if you have special requirements.

19.20.2 Constructors

This is an abstract class, so the constructors are `protected`. You cannot instantiate this class directly—you must use a subclass.

19.20.3 Usage

Numerical axes can be used for the domain and/or range axes in an `XYPlot`, and for the range axis in a `CategoryPlot`.

The methods for obtaining a reference to the axis typically return a `ValueAxis`, so you will need to "cast" the reference to a `NumberAxis` before using any of the methods specific to this class. For example:

```
ValueAxis rangeAxis = myPlot.getRangeAxis();
if (rangeAxis instanceof NumberAxis) {
    NumberAxis axis = (NumberAxis) rangeAxis;
    axis.setAutoRangeIncludesZero(true);
}
```

This casting technique is used often in JFreeChart.

19.20.4 The Axis Range

You can control most aspects of the axis range using methods inherited from the `ValueAxis` class—see section 19.26.5 for details.

Two additional controls are added by this class. First, you can specify whether or not zero must be included in the axis range:

```
axis.setAutoRangeIncludesZero(true);
```

If the *auto-range-includes-zero* flag is set to `true`, then you can further control how the axis margin is calculated when zero falls within the axis margin. By setting the *auto-range-sticky-zero* flag to true:

```
axis.setAutoRangeStickyZero(true);
```

...you can truncate the margin at zero.

19.20.5 Auto Tick Unit Selection

The `NumberAxis` class contains a mechanism for automatically selecting a tick unit from a collection of "standard" tick units. The aim is to display as many ticks as possible, without the tick labels overlapping. The appropriate tick unit will depend on the axis range (which is often a function of the available data) and the amount of space available for displaying the chart.

The *default* standard tick unit collection contains about 50 tick units ranging in size from 0.0000001 to 1,000,000,000. The collection is created and returned by the `createStandardTickUnits(...)` method.

You can replace the default collection with any other collection of tick units you care to create. One common situation where this is necessary is the case where your data consists of integer values only. In this case, you only want the axis to display integer tick values, but sometimes the axis will show values like 0.00, 2.50, 5.00. 7.50, 10.00, when you might prefer 0, 2, 4, 6, 8, 10. For this situation, a set of standard integer tick units has been created. Use the following code:

```
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
TickUnits units = NumberAxis.createIntegerTickUnits();
rangeAxis.setStandardTickUnits(units);
```

For greater control over the tick sizes or formatting, create your own `TickUnits` object.

19.20.6 Attributes

The following table lists the properties maintained by `NumberAxis`, in addition to those inherited from `ValueAxis`.

Attribute:	Description:
auto-range-includes-zero	A flag that indicates whether or not zero is always included when the axis range is determined automatically.
auto-range-sticky-zero	A flag that controls the behaviour of the auto-range calculation when zero falls within the lower or upper margin for the axis. If `true`, the margin will be truncated at zero.
number-format-override	A `NumberFormat` that, if set, overrides the formatting of the tick labels for the axis.

The following default values are used for attributes wherever necessary:

Name:	Value:
DEFAULT_MINIMUM_AXIS_VALUE	0.0
DEFAULT_MAXIMUM_AXIS_VALUE	1.0
DEFAULT_MINIMUM_AUTO_RANGE	new Double(0.0000001);
DEFAULT_TICK_UNIT	new NumberTickUnit(new Double(1.0), new DecimalFormat("0"));

19.20.7 Methods

If you have set the *auto-range* flag to `true` (so that the axis range automatically adjusts to fit the current data), you may also want to set the `AutoRangeIncludesZero` flag to ensure that the axis range always includes zero:

`public void setAutoRangeIncludesZero(boolean flag);`
Sets the *auto-range-includes-zero* flag.

When the *auto-tick-unit-selection* flag is set to `true`, the axis will select a tick unit from a set of standard tick units. You can define your own standard tick units for an axis with the following method:

`public void setStandardTickUnits(TickUnits units);`
Sets the standard tick units for the axis.

You don't have to use the auto tick units mechanism. To specify a fixed tick size (and format):

`public void setTickUnit(NumberTickUnit unit);`
Sets a fixed tick unit for the axis. This allows you to control the size and format of the ticks, but you need to be sure to choose a tick size that doesn't cause the tick labels to overlap.

You can reverse the direction of the values on the axis:

`public void setInverted(boolean flag);`
An *inverted* axis has values that run from high to low, the reverse of the normal case.

19.20.8 Notes

This class defines a default set of standard tick units. You can override the default settings by calling the `setStandardTickUnits(...)` method.

See Also
`ValueAxis`, `TickUnits`.

19.21 NumberTickUnit

19.21.1 Overview

A number tick unit for use by subclasses of `NumberAxis` (extends the `TickUnit` class).

19.21.2 Usage

There are two ways that this class is typically used.

The first is where you know the exact tick size that you want for an axis. In this case, you create a new tick unit then call the `setTickUnit(...)` method in the `ValueAxis` class. For example:

```
XYPlot plot = myChart.getXYPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setTickUnit(new NumberTickUnit(25.0));
```

The second is where you prefer to leave the axis to automatically select a tick unit. In this case, you should create a collection of tick units (see the `TickUnits` class for details).

19.21.3 Constructors

To create a new number tick unit:

`public NumberTickUnit(double size);`
Creates a new number tick unit with a default number formatter for the current locale.

Alternatively, you can supply your own number formatter:

`public NumberTickUnit(double size, NumberFormat formatter);`
Creates a new number tick unit with the specified number formatter.

19.21.4 Methods

To format a value using the tick unit's internal formatter:

`public String valueToString(double value);`
Formats the value as a `String`.

19.21.5 Notes

This class is immutable, a requirement for all subclasses of `TickUnit`.

See Also
`DateTickUnit`.

19.22 SymbolicTickUnit

19.22.1 Overview

Not yet documented.

19.23 Tick

19.23.1 Overview

A utility class representing a tick on an axis. Used temporarily during the drawing process only.

19.23.2 Constructors

The standard constructor:

`public Tick(Object value, String text, float x, float y)`
Creates a tick.

See Also
`TickUnit`.

19.24 TickUnit

19.24.1 Overview

An abstract class representing a tick unit, with subclasses including:

- `DateTickUnit` – for use with subclasses of `DateAxis`;
- `NumberTickUnit` – for use with subclasses of `NumberAxis`.

19.24.2 Constructors

The standard constructor:

```
public TickUnit(double size);
```
Creates a new tick unit with the specified size.

19.24.3 Notes

Implements the `Comparable` interface, so that a collection of tick units can be sorted easily using standard Java methods.

See Also
`TickUnits`.

19.25 TickUnits

19.25.1 Overview

A collection of tick units. This class is used by the `DateAxis` and `NumberAxis` classes to store a list of "standard" tick units. The *auto-tick-unit-selection* mechanism chooses one of the standard tick units in order to maximise the number of ticks displayed without having the tick labels overlap.

19.25.2 Constructors

The default constructor:

```
public TickUnits();
```
Creates a new collection of tick units, initially empty.

19.25.3 Methods

To add a new tick unit to the collection:

```
public void add(TickUnit unit);
```
Adds the tick unit to the collection.

To find the tick unit in the collection that is the next largest in size compared to the specified tick unit:

```
public TickUnit getLargerTickUnit(TickUnit unit);
```
Returns the tick unit that is one size larger than the specified unit.

19.25.4 Notes

The `NumberAxis` class has a static method `createStandardTickUnits()` that generates a tick unit collection (of standard tick sizes) for use by numerical axes.

See Also
`TickUnit`.

19.26 ValueAxis

19.26.1 Overview

The base class for all axes that display "values" (extends `Axis`). Subclasses are divided into those that format the values as `Number` objects (`NumberAxis`) and those that format the values as `Date` objects (`DateAxis`).

At the lowest level, the axis values are manipulated as `double` primitives, obtained from the `Number` objects supplied by the plot's dataset.

19.26.2 Constructors

The constructors for this class are protected, you cannot create a `ValueAxis` directly—you must use a subclass.

19.26.3 Attributes

The attributes maintained by this class, in addition to those that it inherits from the `Axis` class, are listed in Table 5. There are methods to read and update most of these attributes. In general, updating an axis attribute will result in an `AxisChangeEvent` being sent to all (or any) registered listeners.

Attribute:	Description:
anchor-value	Provides a focus point for some operations (for example, zooming).
auto-range	A flag controlling whether or not the axis range is automatically adjusted to fit the range of data values.
auto-tick-unit-selection	A flag controlling whether or not the tick units are selected automatically.
auto-range-minimum-size	The smallest axis range allowed when it is automatically calculated.
lower-margin	The margin to allow at the lower end of the axis scale (expressed as a percentage of the total axis range).
upper-margin	The margin to allow at the upper end of the axis scale (expressed as a percentage of the total axis range).

Table 5: Attributes for the `ValueAxis` class

The default values used to initialise the axis attributes (when necessary) are listed in Table 6.

Name:	Value:
DEFAULT_AUTO_RANGE	true;
DEFAULT_MINIMUM_AXIS_VALUE	0.0;
DEFAULT_MAXIMUM_AXISVALUE	1.0;
DEFAULT_UPPER_MARGIN	0.05 (5 percent)
DEFAULT_LOWER_MARGIN	0.05 (5 percent)

Table 6: ValueAxis class default attribute values

19.26.4 Usage

To modify the attributes of a ValueAxis, you first need to obtain a reference to the axis. For a CategoryPlot, you can use the following code:

```
CategoryPlot plot = myChart.getCategoryPlot();
ValueAxis rangeAxis = plot.getRangeAxis();
// modify the axis here...
```

The code for an XYPlot is very similar, except that the domain axis is also a ValueAxis in this case:

```
XYPlot plot = myChart.getXYPlot();
ValueAxis domainAxis = plot.getDomainAxis();
ValueAxis rangeAxis = plot.getRangeAxis();
// modify the axes here...
```

Having obtained an axis reference, you can:

- control the axis range, see section 19.26.5;

19.26.5 The Axis Range

The *axis range* defines the highest and lowest values that will be displayed on axis. On a chart, it is typically the case that data values outside the axis range are clipped, and therefore not visible on the chart.

By default, JFreeChart is configured to automatically calculate axis ranges so that all of the data in your dataset is visible. It does this by determining the highest and lowest values in your dataset, adding a small margin (to prevent the data being plotted right up to the edge of a chart), and setting the axis range. If you want to, you can turn off this default behaviour, using:

```
axis.setAutoRange(false);
```

You can exercise some control over the auto-range calculation. To set the upper and lower margins (a percentage of the overall axis range):

```
// set margins to 10 percent each...
axis.setLowerMargin(0.10);
axis.setUpperMargin(0.10);
```

19.26.6 Methods

A key function for a `ValueAxis` is to convert a data value to an output (Java2D) coordinate for plotting purposes. The output coordinate will be dependent on the area into which the data is being drawn:

> `public double translateValueToJava2D(double dataValue,`
> `Rectangle2D dataArea);`
> Converts a data value into a co-ordinate along one edge of the `dataArea` (the `dataArea` is the rectangle inside the plot's axes). Whether the coordinate relates to the (left) vertical or (bottom) horizontal edge, depends on the orientation of the axis subclass.

The inverse function converts a Java2D coordinate back to a data value:

> `public double translateJava2DToValue(double java2DValue,`
> `Rectangle2D dataArea);`
> Converts a Java2D coordinate back to a data value.

To control whether or not the axis range is automatically adjusted to fit the available data:

> `public void setAutoRange(boolean auto);`
> Sets a flag (commonly referred to as the *auto-range* flag) that controls whether or not the axis range is automatically adjusted to fit the available data.

To manually set the axis range (which automatically disables the *auto-range* flag):

> `public void setRange(Range range);`
> Sets the axis range.

An alternative method that achieves the same thing:

> `public void setRange(double lower, double upper);`
> Sets the axis range.

To set the lower bound for the axis:

> `public void setMinimumAxisValue(double value);`
> Sets the lower bound for the axis. If the *auto-range* attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

To set the upper bound for the axis:

> `public void setMaximumAxisValue(double value);`
> Sets the upper bound for the axis. If the *auto-range* attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

To set a flag that controls whether or not the axis tick units are automatically selected:

> `public void setAutoTickUnitSelection(boolean flag);`
> Sets a flag (commonly referred to as the *auto-tick-unit-selection* flag) that controls whether or not the tick unit for the axis is automatically selected from a collection of standard tick units.

19.26.7 Notes

In a `CategoryPlot`, the range axis is required to be a subclass of `ValueAxis`.

In an `XYPlot`, both the domain and range axes are required to be a subclass of `ValueAxis`.

See Also

`Axis`, `DateAxis`, `NumberAxis`.

19.27 VerticalAxis

19.27.1 Overview

An interface that *must* be implemented by all vertical axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

19.27.2 Methods

The interface defines two methods—the plot will call one of these two methods at its option.

The first method calculates the width required to display the axis without any knowledge of the height required by the horizontal axis/axes (so an element of guess-work is involved):

> `public double reserveWidth(Graphics2D g2, Plot plot,`
> `Rectangle2D drawArea, int location);`
> Estimates the width that the vertical axis requires to draw itself. If this method is used, it will be called *before* the horizontal axis is asked to calculate the height that it requires—and the width returned by this method will be passed to the horizontal axis when it is asked to calculate the height it requires.

The second method is similar to the first except that the height required for the horizontal axis/axes has already been calculated:

> `public double reserveWidth(Graphics2D g2, Plot plot, Rectangle2D drawArea,`
> `int location, double reservedHeight, int horizontalAxisLocation);`
> Calculates the width that the vertical axis requires to draw itself. If this method is used, it will be called *after* the horizontal axis has estimated the height that it requires—the argument `reservedHeight` contains this value.

19.27.3 Notes

For horizontal axes, the `HorizontalAxis` interface performs a similar role.

19.28 VerticalCategoryAxis

19.28.1 Overview

A vertical axis that displays categories—extends `CategoryAxis` and implements `VerticalAxis`.

This axis is used with the `HorizontalCategoryPlot` class.

19.28.2 Constructor

To create a new axis:

`public VerticalCategoryAxis(String label);`
Creates a new axis, with the supplied label (`null` permitted).

The axis will be initialised using default values, you can subsequently change these if required.

19.28.3 Attributes

The `VerticalCategoryAxis` class maintains the following attributes, in addition to those it inherits from `CategoryAxis` (refer to section 19.5.3 for details):

Attribute:	Description:
vertical-label	A flag that controls whether the axis label is rotated to a "vertical" orientation.

The following default values are used:

Default:	Value:
`DEFAULT_VERTICAL_LABEL`	`true`

See Also

`HorizontalCategoryAxis`.

19.29 VerticalColorBarAxis

19.29.1 Overview

Not yet documented.

19.30 VerticalDateAxis

19.30.1 Overview

An axis that displays *date/time* values—extends `DateAxis` and implements `VerticalAxis`.

With its vertical orientation, this axis can be used as the range axis for an `XYPlot` or a `VerticalCategoryPlot`.

19.30.2 Constructors

There are several constructors available, the most commonly used is:

```
public VerticalDateAxis(String label);
```
Creates a new axis with the specified label (`null` permitted).

Refer to the Javadoc HTML files for information about the other constructors.

19.30.3 Attributes

This class adds the following attributes to those it inherits from the `DateAxis` class:

Attribute:	Description:
vertical-label	A flag that controls whether or not the axis label is displayed "vertically" (that is, rotated 90 degrees from horizontal).

Refer to section 19.7.3 for information about the attributes inherited by this class.

19.30.4 Usage

To change the attributes of the axis, you need to obtain a `VerticalDateAxis` reference—because of the way JFreeChart is designed, this usually involves a "cast":

```
CategoryPlot plot = myChart.getCategoryPlot();
ValueAxis rangeAxis = plot.getRangeAxis();
if (rangeAxis instanceof VerticalDateAxis) {
    VerticalDateAxis axis = (VerticalDateAxis) domainAxis;
    // customise axis here...
}
```

Given a `VerticalDateAxis` reference, you can change:

- the orientation of the axis label, see section 19.30.5;
- general setting (the range, tick size and formatting, etc.), see section 19.7.4.

19.30.5 Axis Label Orientation

To control the orientation of the axis label:

```
axis.setVerticalLabel(true);
```

Code similar to this can be used for all vertical axes.

19.30.6 Notes

Although the axis displays dates for tick labels, at the lowest level it is still working with `double` primitives obtained from the `Number` objects supplied by the plot's dataset. The values are interpreted as *the number of milliseconds since 1 January 1970* (that is, the same encoding used by `java.util.Date`).

See Also

HorizontalDateAxis.

19.31 VerticalLogarithmicAxis

19.31.1 Overview

A numerical axis that displays values using a logarithmic scale.

19.31.2 Notes

An equivalent class `HorizontalLogarithmicAxis` has now been implemented.

See Also

NumberAxis.

19.32 VerticalLogarithmicColorBarAxis

19.32.1 Overview

Not yet documented.

19.33 VerticalNumberAxis

19.33.1 Overview

A vertical axis that displays numerical data—this class extends `NumberAxis`.

19.33.2 Constructors

There is a single constructor for this class:

`public VerticalNumberAxis(String label)`
Creates a new axis with the specified label (`null` permitted).

19.33.3 Methods

A list of important methods:

`public void autoAdjustRange();`
This method obtains the maximum and minimum data values from the `Plot`, provided that it implements `VerticalValueRange`, and adjusts the axis range accordingly. Note that the *auto-range-includes-zero* flag is checked in this method.

`public void refreshTicks(...);`
A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the *auto-tick-units* flag, and automatically determines a suitable "standard" tick size if required.

See Also

NumberAxis, HorizontalNumberAxis.

19.34 VerticalNumberAxis3D

19.34.1 Overview

A vertical axis that draws itself with a 3D-effect. In all other respects, the axis should behave in the same way as the `VerticalNumberAxis` class.

19.35 VerticalSymbolicAxis

19.35.1 Overview

A numerical axis that displays values using symbols.

See Also

`NumberAxis`.

20 Package: org.jfree.chart.entity

20.1 Introduction

The `org.jfree.chart.entity` package contains classes that represent entities in a chart.

20.2 Background

Recall that when you render a chart to a `Graphics2D` using the `draw(...)` method in the `JFreeChart` class, you have the option of supplying a `ChartRenderingInfo` object to collect information about the chart's dimensions. Most of this information is represented in the form of `ChartEntity` objects, stored in an `EntityCollection`.

You can use the entity information in any way you choose. For example, the `ChartPanel` class makes use of the information for:

- displaying tool tips;
- handling chart mouse events.

It is more than likely that other applications for this information will be found.

20.3 CategoryItemEntity

20.3.1 Overview

This class is used to convey information about an item within a category plot. The information captured includes the area occupied by the item, the tool tip text generated for the item, and the series and category that the item represents.

20.3.2 Constructors

To construct a new instance:

> `public CategoryItemEntity(Shape area, String toolTipText, int series, Object category);`
> Creates a new entity instance.

20.3.3 Methods

Accessor methods are implemented for the `series` and `category` attributes. Other methods are inherited from the `ChartEntity` class.

You can generate an `AREA` tag for an HTML image map:

> `public String getImageMapAreaTag(String hrefPrefix);`
> Returns a tag that can be used to represent this entity when creating an HTML image map.

20.3.4 Notes

Most `CategoryItemRenderer` implementations will generate entities using this class, as required.

See Also

ChartEntity, CategoryPlot.

20.4 ChartEntity

20.4.1 Overview

This class is used to convey information about an entity within a chart. The information captured includes the area occupied by the item and the tool tip text generated for the item.

There are a number of subclasses that can be used to provide additional information about a chart entity.

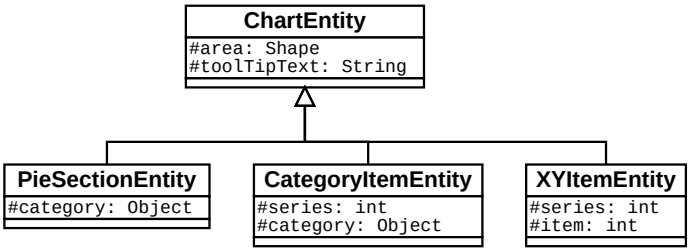


Figure 4: Chart entity classes

20.4.2 Constructors

To construct a new instance:

`public ChartEntity(Shape area, String toolTipText);`
Creates a new chart entity object. The area is specified in Java 2D space.

Chart entities are created by other classes in the JFreeChart library, you don't usually need to create them yourself.

20.4.3 Methods

Accessor methods are implemented for the `area` and `toolTipText` attributes.

To support the generation of HTML image maps, the `getShapeType()` method returns a `String` containing either `RECT` or `POLY`, and the `getShapeCoords()` method returns a `String` containing the coordinates of the shape's outline. See the `ChartUtilities` class for more information about HTML image maps.

20.4.4 Notes

The `ChartEntity` class *records* where an entity has been drawn using a `Graphics-2D` instance. Changing the attributes of an entity won't change what has already been drawn.

See Also

CategoryItemEntity, PieSectionEntity, XYItemEntity.

20.5 ContourEntity

20.5.1 Overview

Not yet documented.

20.6 EntityCollection

20.6.1 Overview

An interface that defines the API for a collection of *chart entities*. This is used by the `ChartRenderingInfo` class to record where items have been drawn when a chart is rendered using a `Graphics2D` instance.

Each `ChartEntity` can also record tool tip information (for displaying tool tips in a Swing user interface) and/or URL information (for generating HTML image maps).

20.6.2 Methods

The interface defines three methods. To clear a collection:

```
public void clear();
```
Clears the collection. All entities in the collection are discarded.

To add an entity to a collection:

```
public void addEntity(ChartEntity entity);
```
Adds an entity to the collection.

To retrieve an entity based on Java 2D coordinates:

```
public ChartEntity getEntity(double x, double y);
```
Returns an entity whose area contains the specified coordinates. If the coordinates fall within the area of multiple entities (the entities overlap) then only one entity is returned.

20.6.3 Notes

The `StandardEntityCollection` class provides a basic implementation of this interface (but one that won't scale to large numbers of entities).

See Also

`ChartEntity`, `StandardEntityCollection`.

20.7 PieSectionEntity

20.7.1 Overview

This class is used to convey information about an item within a pie plot. The information captured includes the area occupied by the item, the tool tip text generated for the item and the category that the item represents.

20.7.2 Constructors

To construct a new instance:

```
public PieSectionEntity(Shape area, String toolTipText, Object category);
```
Creates a new entity object.

20.7.3 Methods

Accessor methods are implemented for the `category` attribute. Other methods are inherited from the `ChartEntity` class.

20.7.4 Notes

The `PiePlot` class generates pie section entities as required.

See Also

`ChartEntity`, `PiePlot`.

20.8 StandardEntityCollection

20.8.1 Overview

A basic implementation of the `EntityCollection` interface. This class can be used (optionally, by the `ChartRenderingInfo` class) to store a collection of chart entity objects from one rendering of a chart.

20.8.2 Methods

This class implements the methods in the `EntityCollection` interface.

20.8.3 Notes

The `getEntity(...)` method iterates through the entities searching for one that contains the specified coordinates. For charts with a large number of entities, a more efficient approach will be required.[7]

See Also

`ChartEntity`, `EntityCollection`.

20.9 XYItemEntity

20.9.1 Overview

This class is used to convey information about an item within an XY plot. The information captured includes the area occupied by the item, the tool tip text generated for the item, and the series and item index.

[7] This is on the to-do list but, given the size of the to-do list, I'm hopeful that someone will contribute code to address this.

20.9.2 Constructors

To construct a new instance:

`public XYItemEntity(Shape area, String toolTipText, int series, int item);`
Creates a new entity object.

20.9.3 Methods

Accessor methods are implemented for the `series` and `item` attributes. Other methods are inherited from the `ChartEntity` class.

20.9.4 Notes

Most `XYItemRenderer` implementations will generate entities using this class, as required.

See Also

`ChartEntity`, `XYPlot`.

21 Package: org.jfree.chart.event

21.1 Introduction

This package contains classes and interfaces that are used to broadcast and receive events relating to changes in chart properties. By default, some of the classes in the library will automatically register themselves with other classes, so that they receive notification of any changes and can react accordingly. For the most part, you can simply rely on this default behaviour.

21.2 AxisChangeEvent

21.2.1 Overview

An event that is used to provide information about changes to axes.

See Also

AxisChangeListener.

21.3 AxisChangeListener

21.3.1 Overview

An interface through which axis change event notifications are posted.

21.3.2 Methods

The interface defines a single method:

```
public void axisChanged(AxisChangeEvent event);
```

Receives notification of a change to an axis.

21.3.3 Notes

If a class needs to receive notification of changes to an axis, then it needs to implement this interface and register itself with the axis.

See Also

AxisChangeEvent.

21.4 ChartChangeEvent

21.4.1 Overview

An event that is used to provide information about changes to a chart.

See Also

ChartChangeListener.

21.5 ChartChangeListener

21.5.1 Overview

An interface through which chart change event notifications are posted.

21.5.2 Methods

The interface defines a single method:

```
public void chartChanged(ChartChangeEvent event);
```
Receives notification of a change to a chart.

21.5.3 Notes

If a class needs to receive notification of changes to a chart, then it needs to implement this interface and register itself with the chart.

See Also
ChartChangeEvent.

21.6 ChartProgressEvent

21.6.1 Overview

Not yet documented.

21.7 ChartProgressListener

21.7.1 Overview

Not yet documented.

21.8 LegendChangeEvent

21.8.1 Overview

An event that is used to provide information about changes to a legend.

See Also
LegendChangeListener.

21.9 LegendChangeListener

21.9.1 Overview

An interface through which legend change event notifications are posted.

21.9.2 Methods

The interface defines a single method:

```
public void legendChanged(LegendChangeEvent event);
```
Receives notification of a change to a legend.

21.9.3 Notes

If a class needs to receive notification of changes to a legend, then it needs to implement this interface and register itself with the legend.

See Also
LegendChangeEvent.

21.10 PlotChangeEvent

21.10.1 Overview

An event that is used to provide information about changes to a plot.

See Also
PlotChangeListener.

21.11 PlotChangeListener

21.11.1 Overview

An interface through which plot change event notifications are posted.

21.11.2 Methods

The interface defines a single method:

```
public void plotChanged(PlotChangeEvent event);
```
Receives notification of a change to a plot.

21.11.3 Notes

If a class needs to receive notification of changes to a plot, then it needs to implement this interface and register itself with the plot.

See Also
PlotChangeEvent.

21.12 TitleChangeEvent

21.12.1 Overview

An event that is used to provide information about changes to a chart title (any subclass of `AbstractTitle`).

21.12.2 Notes

This event is part of the overall mechanism that JFreeChart uses to automatically update charts whenever changes are made to components of the chart.

See Also
AbstractTitle, TitleChangeListener.

21.13 TitleChangeListener

21.13.1 Overview

An interface through which title change event notifications are posted.

21.13.2 Methods

The interface defines a single method:

```
public void titleChanged(TitleChangeEvent event);
```
Receives notification of a change to a title.

21.13.3 Notes

If a class needs to receive notification of changes to a title, then it needs to implement this interface and register itself with the title.

See Also

`TitleChangeEvent`.

22 Package: org.jfree.chart.needle

22.1 Overview

This package contains classes for drawing needles in a compass plot.

22.2 ArrowNeedle

22.2.1 Overview

Not yet documented.

22.3 LineNeedle

22.3.1 Overview

Not yet documented.

22.4 LongNeedle

22.4.1 Overview

Not yet documented.

22.5 MeterNeedle

22.5.1 Overview

Not yet documented.

22.6 PinNeedle

22.6.1 Overview

Not yet documented.

22.7 PlumNeedle

22.7.1 Overview

Not yet documented.

22.8 PointerNeedle

22.8.1 Overview

Not yet documented.

22.9 ShipNeedle

22.9.1 Overview

Not yet documented.

22.10 WindNeedle

22.10.1 Overview

Not yet documented.

23 Package: org.jfree.chart.plot

23.1 Overview

The `org.jfree.chart.plot` package contains:

- the `Plot` base class;
- a range of plot subclasses;
- various support classes and interfaces.

This is an important package, because the `Plot` classes play a key role in controlling the presentation of data with JFreeChart.

23.2 CategoryPlot

23.2.1 Overview

A general plotting class that is most commonly used to display bar charts, but also supports line charts, area charts, stacked area charts and more.

- the plot uses a `CategoryAxis` for its *domain axis*, and a `ValueAxis` for its *range axis*;
- data for the plot is obtained via the `CategoryDataset` interface, and rendered using a `CategoryItemRenderer`;
- an optional *secondary dataset* and *secondary range axis* is supported (since version 0.9.5);
- two different orientations of the plot are possible, implemented by the classes `HorizontalCategoryPlot` and `VerticalCategoryPlot`.

23.2.2 Constructors

This class is abstract, so the constructors are `protected`. You cannot create an instance of this class directly, you must use a subclass.

23.2.3 Attributes

The attributes maintained by the `CategoryPlot` class, which are in addition to those inherited from the `Plot` class, are listed in Table 7.

23.2.4 Axes

The plot's domain axis is an instance of `CategoryAxis`. You can obtain a reference to the axis with:

```
CategoryAxis axis = myPlot.getDomainAxis();
```

The plot's range axis is an instance of `ValueAxis`. You can obtain a reference to the axis with:

```
ValueAxis axis = myPlot.getRangeAxis();
```

The axis classes have many attributes that can be customised to control the appearance of your charts.

Attribute:	Description:
domain-axis	The domain axis (used to display categories).
domain-axis-location	The location of the domain axis.
range-axis	The range axis (used to display values).
range-axis-location	The location of the range axis.
renderer	The plot's renderer (a "pluggable" object responsible for drawing individual data items within the plot).
domain-gridlines-visible	A flag that controls whether gridlines are drawn against the domain axis.
domain-gridline-paint	The paint used to draw the domain gridlines.
domain-gridline-stroke	The stroke used to draw the domain gridlines.
range-gridlines-visible	A flag that controls whether gridlines are drawn against the range axis.
range-gridline-paint	The paint used to draw the range gridlines.
range-gridline-stroke	The stroke used to draw the range gridlines.
range-markers	A list of markers (constants) to be highlighted on the plot.
value-labels-visible	A flag controlling whether or not value labels are drawn on the plot (some renderers observe this setting, others do not).
value-label-font	The font used to draw value labels.
value-label-paint	The paint used when drawing value labels.
vertical-value-labels	A flag that controls whether or not the value labels are drawn vertically.
secondary-dataset	The secondary dataset (optional).
secondary-range-axis	The secondary range axis (optional).
secondary-renderer	The renderer for the secondary dataset. If `null`, the primary renderer will be used.

Table 7: Attributes for the `CategoryPlot` class

23.2.5 Series Colors

The colors used for the series within the chart are controlled by the plot's *renderer*. You can obtain a reference to the renderer using:

```
CategoryPlot plot = myChart.getCategoryPlot();
AbstractRenderer renderer = (AbstractRenderer) plot.getRenderer();
```

23.2.6 Gridlines

By default, the `CategoryPlot` class will display gridlines against the range axis, but not the domain axis. However, it is simple to override the default behaviour:

```
CategoryPlot plot = myChart.getCategoryPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);
```

Note that the domain and range gridlines are controlled independently.

23.2.7 Methods

You can control the appearance of the plot by setting a renderer for the plot. The renderer is responsible for drawing a visual representation of each data item:

```
public void setRenderer(CategoryItemRenderer renderer);
```

Sets the renderer for the plot. A range of different renderers are available.

If you set the renderer to `null`, an empty chart is drawn.

To get a reference to the category axis for the plot:

```
public abstract CategoryAxis getDomainAxis();
```
Returns the category axis for the plot.

To get a reference to the numerical axis for the plot:

```
public abstract ValueAxis getRangeAxis();
```
Returns the value axis for the plot.

A zoom method is provided to support the zooming function provided by the `ChartPanel` class:

```
public void zoom(double percent);
```
Increases or decreases the axis range (about the anchor value) by the specified percentage. If the percentage is zero, then the auto-range calculation is restored for the value axis.

The category axis remains fixed during zooming, only the value axis changes.

Some renderers support the display of "value labels" next to each item on the chart. To switch this feature on:

```
public void setValueLabelsVisible(boolean flag);
```
Sets the flag that controls whether or not value labels are displayed for the items in the plot. Not all renderers support this.

23.2.8 Notes

A number of `CategoryItemRenderer` implementations are included in the JFreeChart distribution.

Not all of the renderers recognise the *value-labels-visible* flag yet. For those that do, the positioning of the labels is fixed, there are no attributes to control the positioning. Future versions of JFreeChart will probably include enhancements in this area.

See Also

`HorizontalCategoryPlot`, `VerticalCategoryPlot`.

23.3 CategoryPlotConstants

23.3.1 Overview

An interface that defines constants used by the `CategoryPlot` class.

23.4 CombinedXYPlot

23.4.1 Overview

A subclass of `XYPlot` that allows you to combined multiple plots on one chart. The layout of the subplots can be `HORIZONTAL` or `VERTICAL`. The subplots share either the domain (horizontal) or range (vertical) axis from the `CombinedXYPlot` (depending on the layout), and maintain one "non-shared" axis each.

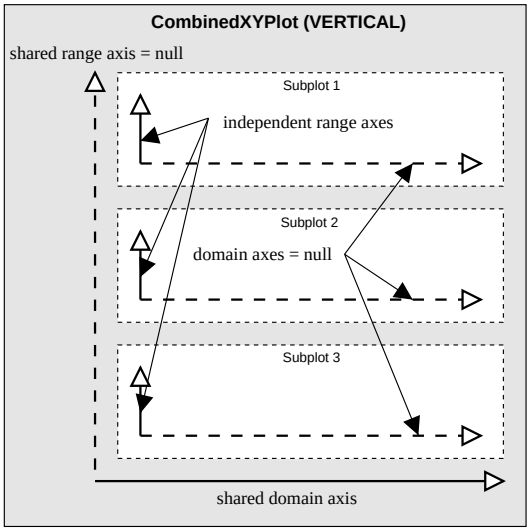


Figure 5: CombinedXYPlot axes

Figure 5 illustrates the relationship between the `CombinedXYPlot` and its subplots (in this case the combination is `VERTICAL`).

The `CombinedXYPlotDemo` class provides an example of this type of plot.

23.4.2 Methods

There are two methods for adding a subplot to a combined plot:

`public void add(XYPlot subplot);`
Adds a subplot to the combined plot, with a weight of `1`.

`public void add(XYPlot subplot, int weight);`
Adds a subplot to the combined plot, with the specified weight.

The subplot being added to the `CombinedXYPlot` can be any instance of `XYPlot` (including an `OverlaidXYPlot`) and should have one of its axes (the shared axis) set to `null`.

The weight determines how much of the plot area is assigned to the subplot. For example, if you add three subplots with weights of 1, 2 and 4, the relative amount of space assigned to each plot is 1/7, 2/7 and 4/7 (where the 7 is the sum of the individual weights).

To control the amount of space between the subplots:

`public void setGap(double gap);`
Sets the gap (in points) between the subplots.

23.4.3 Notes

The dataset for this class should be set to `null` (only the subplots display data).

The subplots managed by this class should have one axis set to `null` (the shared axis is maintained by this class).

You do not need to set a renderer for the plot, since each subplot maintains its own renderer.

Each subplot uses its own series colors. You should modify the default colors to ensure that the items for each subplot are uniquely colored.

A demonstration of this type of plot is described in section 9.3.

See Also

`XYPlot`, `OverlaidXYPlot`.

23.5 CompassPlot

23.5.1 Overview

A *compass plot* presents directional data in the form of a compass dial.

23.5.2 Notes

The `CompassDemo` application (included in the JFreeChart distribution) demonstrates the use of this class.

23.6 ContourPlot

23.6.1 Overview

A custom plot that displays *(x, y, z)* data in the form of a 2D contour plot.

23.7 ContourPlotUtilities

23.7.1 Overview

A class that contains static utility methods used by the contour plot implementation.

23.8 ContourValuePlot

23.8.1 Overview

An interface used by the contour plot implementation.

23.9 FastScatterPlot

23.9.1 Overview

A class that plots data directly from an array, rather than using the `XYDataset` interface.

23.9.2 Notes

This class is provided as an illustration of how you can by-pass the dataset interfaces if you do not like the overhead that those interfaces add to the rendering process.

23.10 HorizontalCategoryPlot

23.10.1 Overview

This plot draws a chart using data from a `CategoryDataset`, where the categories are plotted against the vertical axis and the numerical data is plotted against the horizontal axis.

23.10.2 Constructors

This class provides two constructors—one that requires all the attributes for the plot to be specified, the other assumes a number of default values. Refer to the Javadoc or the source code for details.

23.10.3 Notes

This class inherits most of its functions from the `CategoryPlot` class.

See Also

`VerticalCategoryPlot`.

23.11 HorizontalValuePlot

23.11.1 Overview

An interface that returns the range of values in the "horizontal direction" for a two-dimensional plot. The values could be from the dataset's domain or range, depending on the orientation of the plot.

23.11.2 Methods

To get the range:

```
public Range getHorizontalDataRange(ValueAxis axis);
```

Returns range of data values to be plotted against the horizontal axis.

To get the axis:

```
public ValueAxis getHorizontalValueAxis();
```

Returns the horizontal value axis.

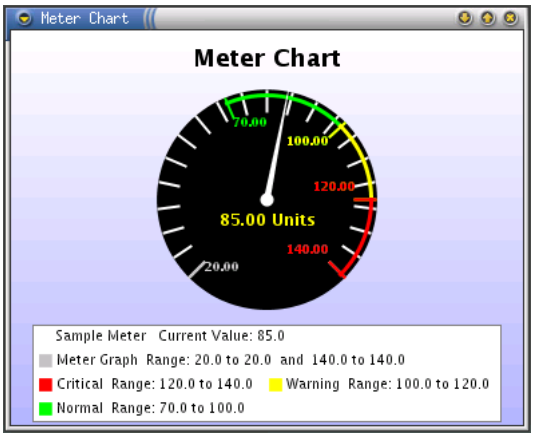
See Also

`VerticalValuePlot`.

23.12 MeterPlot

23.12.1 Overview

A plot that displays a single value in a dial presentation. The current value is represented by a needle in the dial, and is also displayed in the center of the dial in text format.



Three ranges on the dial provide some context for the value: the *normal range*, the *warning range* and the *critical range*.

23.12.2 Constructors

To create a new `MeterPlot`:

`public MeterPlot(MeterDataset data);`
Creates a dial with default settings, using the supplied dataset.

If you want to have more control over the appearance of the dial:

`public MeterPlot(MeterDataset data, Insets insets, Paint backgroundPaint, Image backgroundImage, float backgroundAlpha, Stroke outlineStroke, Paint outlinePaint, float foregroundAlpha, int tickLabelType, Font tickLabelfont);`
Creates a dial with the supplied settings and dataset.

23.12.3 Methods

A needle is used to indicate the current value on the dial. To change the color of the needle:

`public void setNeedlePaint(Paint paint);`
Sets the color of the needle on the dial. The default is `Color.green`. If you pass in `null` to this method, the needle color reverts to the default.

The current value is also displayed (near the center of the dial) in text format. To change the font used to display the current value:

`public void setValueFont(Font font);`
Sets the font used to display the current value.

To change the color used to display the current value:

`public void setValuePaint(Paint paint);`
Sets the paint used to display the current value.

To change the background color of the dial:

`public void setDialBackgroundPaint(Paint paint);`
Sets the color of the dial background. The default is `Color.black`. If you pass in `null` to this method, the background color reverts to the default.

By default, the needle on the dial is free to rotate through 270 degrees (centered at 12 o'clock). To change this, use this method:

`public void setMeterAngle(int angle);`
Sets the range within which the dial's needle can move.

Related to the above is the shape of the dial: circular (the default), pie or chord:

`public void setDialType(int type);`
Sets the shape of the dial. The default is `DIALTYPE_CIRCLE`. The other options are `DIALTYPE_PIE` and `DIALTYPE_CHORD`.

The three context ranges are drawn as color highlights near the outer edge of the dial. To change the highlight color of the normal range:

`public void setNormalPaint(Paint paint);`
Sets the color of the normal range. The default is `Color.green`. If you pass in `null` to this method, the color reverts to the default.

To change the highlight color of the warning range:

`public void setWarningPaint(Paint paint);`
Sets the color of the warning range. The default is `Color.yellow`. If you pass in `null` to this method, the color reverts to the default.

To change the highlight color of the critical range:

`public void setCriticalPaint(Paint paint);`
Sets the color of the critical range. The default is `Color.red`. If you pass in `null` to this method, the color reverts to the default.

To control whether or not labels are displayed for the values in the normal, warning, critical and overall ranges:

`public void setTickLabelType(int type);`
Controls whether or not tick labels are displayed. The `type` should be one of: `NO_LABELS` and `VALUE_LABELS`.

If tick labels are displayed, the font can be set using:

`public void setTickLabelFont(Font font);`
Sets the font used to display tick labels (if they are visible).

23.12.4 Notes

This chart type was contributed by Hari.

The `MeterPlotDemo` class in the `com.jrefinery.chart.demo` package provides a working example of this class.

In the current version, a fixed number of ticks (20) are drawn for the dial range, irrespective of the maximum and minimum data values. The tick generation will be enhanced in a future release.

See Also

`MeterDataset`, `MeterLegend`.

23.13 OverlaidVerticalCategoryPlot

23.13.1 Overview

A *vertical category plot* that allows multiple subplots to share a common set of axes.

23.13.2 Methods

To add a new subplot:

> `public void add(VerticalCategoryPlot subplot);`
> Adds a subplot. The subplot's axes will be set to `null`, to ensure that it uses the shared axes.

The following method returns the dataset from the first subplot. It is used by the domain axis to determine the categories to display:

> `public CategoryDataset getCategoryDataset();`
> Returns the dataset from the first subplot.

All the datasets from the remaining subplots are required to use the same categories.

Items for the legend are collected from all the subplots:

> `public LegendItemCollection getLegendItems();`
> Returns a list of all the series in all the subplots.

The vertical data range is calculated by merging the ranges of all the subplots:

> `public Range getVerticalDataRange(ValueAxis axis);`
> Returns the combined range of all the subplots.

23.13.3 Notes

All of the datasets in the subplots must use the same set of categories. Unexpected results will occur if this is not the case.

The `OverlaidBarChartDemo` class, included in the JFreeChart distribution, illustrates the use of this class.

23.14 OverlaidXYPlot

23.14.1 Overview

A subclass of `XYPlot`, this class allows you to combine multiple subplots within a single chart. As far as possible, this class tries to behave in exactly the same way as a regular `XYPlot`. Setting axis ranges, background colors and so forth should be no different to usual.

One important difference between this class and `XYPlot` is that you do not supply a dataset for overlaid plots. Each of the subplots maintains its own dataset.

All the subplots (instances of `XYPlot`) should have `null` axes, because they share the axes managed by the `OverlaidXYPlot`. When you set the properties of an axis belonging to an overlaid plot (the *parent plot*) all of the subplots will update to reflect the change.

Figure 6 illustrates the relationship between the parent plot and its subplots.

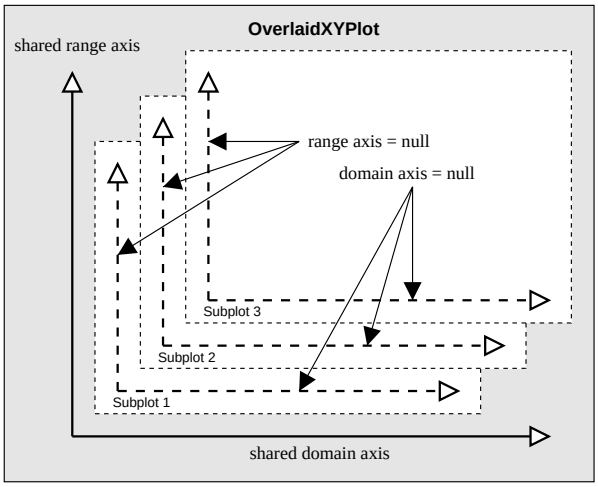


Figure 6: OverlaidXYPlot axes

23.14.2 Constructors

To construct a new `OverlaidXYPlot`:

> `public OverlaidXYPlot(ValueAxis domain, ValueAxis range);`
> Creates a new plot with the specified axes. No dataset is necessary, since the subplots (which you must add) maintain their own datasets.

Another constructor, which takes a domain axis label and a range axis label as arguments, creates a new plot with numerical axes. This is provided for convenience, allowing you to construct a new plot without having to first construct axes.

23.14.3 Methods

To add a subplot:

`public void add(XYPlot subplot);`
Adds a subplot. The subplot can be almost any instance of `XYPlot` and should have both its axes set to `null`.

23.14.4 Notes

The dataset for this class should be set to `null`.

The subplots managed by this class should have their domain and range axes set to `null`.

A demonstration of this type of plot is described in section 9.2.

See Also
`XYPlot`, `CombinedXYPlot`.

23.15 PeriodMarkerPlot

23.15.1 Overview

A plot that highlights time periods using different colors.

23.15.2 Notes

This was contributed by Sylvain Vieujot. I haven't done any work with this class yet, but my initial thought is that it could be converted to an `XYItemRenderer`.

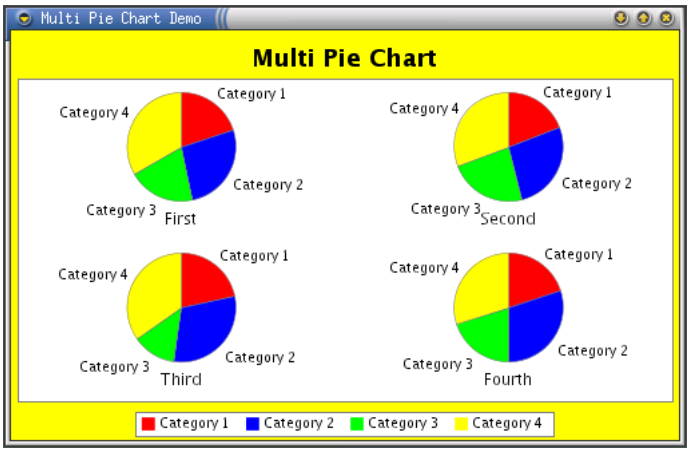
See Also
`XYPlot`.

23.16 PiePlot

23.16.1 Overview

The `PiePlot` class draws pie charts, using data obtained through the `PieDataset` interface (for a single pie chart) or the `CategoryDataset` interface (for multiple pie charts).

Here is an example of a pie chart based on a `CategoryDataset`:



A related class, `Pie3DPlot`, draws pie charts with a 3D effect.

23.16.2 Constructors

To construct a pie plot:

`public PiePlot(PieDataset data);`
Creates a pie plot with default attributes.

To construct a "multi" pie plot:

`public PiePlot(CategoryDataset data);`
Creates a pie plot that displays one pie chart for each series in the dataset.

Finally, there is a complex constructor that allows you to specify all the attributes for the plot. Refer to the Javadocs or source code for details.

23.16.3 Attributes

The attributes maintained by the `PiePlot` class, which are in addition to those inherited from the `Plot` class, are listed in Table 8.

The following default values are used where necessary:

Name:	Value:
DEFAULT_INTERIOR_GAP	0.20 (20 percent)
DEFAULT_RADIUS	1.00 (100 percent)
DEFAULT_SECTION_LABEL_FONT	new Font("SansSerif", Font.PLAIN, 10);
DEFAULT_SECTION_LABEL_PAINT	Color.black;
DEFAULT_SECTION_LABEL_GAP	0.10 (10 percent)

23.16.4 Methods

To control whether the pie chart is circular or elliptical:

`public void setCircular(boolean flag);`
Sets a flag that controls whether the pie chart is circular or elliptical in shape.

Attribute:	Description:
interior-gap	The space to leave blank around the outside of the pie.
circular	Circular or elliptical pie.
radius	Controls the radius of the unexploded pie.
start-angle	The angle of the first pie section. Zero degrees is three o'clock, 90 degrees is twelve o'clock, 180 degrees is nine o'clock and 270 degrees is six o'clock.
direction	Pie sections can be ordered in a clockwise or anticlockwise direction.
section-label-type	The type of labels for the pie sections.
section-label-font	The font for the section labels.
section-label-paint	The color for the section labels.
section-label-gap	The gap for the section labels.
explode-percentages[]	The amount to 'explode' each pie section.
tooltip-generator	A plug-in tool tip generator.
url-generator	A plug-in URL generator (for image map generation).

Table 8: Attributes for the `PiePlot` class

To control the direction (clockwise or anticlockwise) of the sections in the pie chart:

`public void setDirection(int direction);`
Sets the direction of the sections in the pie chart. Use one of the constants `CLOCKWISE` (the default) and `ANTICLOCKWISE`.

To control the position of the first section in the chart:

`public void setStartAngle(double angle);`
Defines the angle (in degrees) at which the first section starts. Zero is at 3 o'clock, and as the angle increases it proceeds anticlockwise around the chart (so that 90 degrees, the current default, is at 12 o'clock).

To control the amount of space around the pie chart:

`public void setInteriorGapPercent(double percent);`
Sets the amount of space inside the plot area. This space is used for displaying section labels and, currently, there is no automated checking to ensure that there is sufficient space to display the labels.

To control the style of the labels for each section of the pie chart:

`public void setSectionLabelType(int type);`
Sets the type of label to display next to each section of the pie chart. Use one of the following constants: `NO_LABELS`, `NAME_LABELS`, `VALUE_LABELS`, `PERCENT_LABELS`, `NAME_AND_VALUE_LABELS`, `NAME_AND_PERCENT_LABELS` and `VALUE_AND_PERCENT_LABELS`.

A pie plot is drawn with this method:

`public void draw(Graphics2D g2, Rectangle2D drawArea, ChartRenderingInfo info);`
Draws the pie plot within the specified drawing area. Typically, this method will be called for you by the `JFreeChart` class.

The `info` parameter is optional. If you pass in an instance of `ChartRenderingInfo`, it will be populated with information about the chart (for example, chart dimensions and tool tip information).

To set the tooltip generator (optional) for the pie plot:

`public void setToolTipGenerator(PieToolTipGenerator generator);`
Registers a tooltip generator with the pie plot. If you write your own generator, you can have full control over the tooltip text that is generated for each pie section.

23.16.5 Exploded Sections

It is possible to "explode" sections of the pie chart, although the current implementation is not as clean as it might be. Hopefully the following explanation will help to clarify how it works.

By default, the "pie" will occupy the largest circle (or ellipse) available to it (this will be affected by the *interior gap percent* setting, which provides a crude mechanism for making more room available for longer labels). If you want to explode any sections in the chart, then you need to shrink the pie to make room. You can use the `setRadius(...)` method for this.

If you use a setting of (say) `0.60` (60 percent), this defines an *inner circle* with a radius of 60 percent of the maximum (the *outer circle*). The *unexploded* pie sections will be drawn within this inner circle.

Now, consider the gap between the inner circle and the outer circle. For each *exploded* pie section, you specify a percentage value (using the `setExplodePercent(...)` method) which controls how far towards the outer circle the section is "exploded". 100 percent means the section will touch the outer circle, 50 percent means it will be half way between the inner circle and the outer circle, and so on.

The `PieChartDemo2` application (included in the JFreeChart distribution) provides a demo.

23.16.6 Notes

There are several methods in the `ChartFactory` class that will construct a default pie chart for you.

The `DatasetUtilities` class has methods for creating a `PieDataset` from a `CategoryDataset`.

The `PieChartDemo1` class in the `com.jrefinery.chart.demo` package provides a simple pie chart demonstration.

See Also

`PieDataset`, `PieToolTipGenerator`, `Plot`.

23.17 Pie3DPlot

23.17.1 Overview

An extension of the `PiePlot` class that draws pie charts with a 3D effect.

23.18 Plot

23.18.1 Overview

An abstract base class that controls the visual representation of data in a chart. The `JFreeChart` class maintains a reference to a `Plot`, and will provide it with an area in which to draw itself (after allocating space for the chart titles and legend).

A range of subclasses are used to create different types of charts:

- `CategoryPlot` – for bar charts and other plots where one axis displays categories and the other axis displays values;
- `MeterPlot` – dials, thermometers and other plots that display a single value;
- `PiePlot` – for pie charts;
- `XYPlot` – for line charts, scatter plots, time series charts and other plots where both axes display numerical (or date) values;

Figure 7 illustrates the plot class hierarchy.

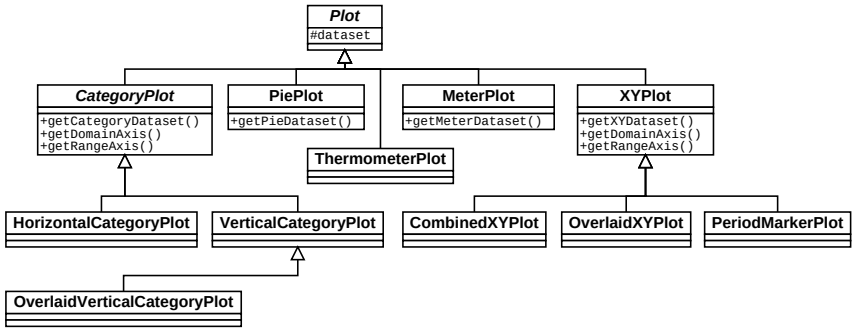


Figure 7: Plot classes

When a chart is drawn, the `JFreeChart` class first draws the title (or titles) and legend. Next, the plot is given an area (the *plot area*) into which it must draw a representation of its dataset. This function is implemented in the `draw(...)` method, each subclass of `Plot` takes a slightly different approach.

23.18.2 Constructors

This class is abstract, so the constructors are

23.18.3 Attributes

This class maintains the following attributes:

Attribute:	Description:
dataset	The primary dataset.
secondary-dataset	The secondary dataset (optional).
dataset-group	The dataset group (to be used for synchronising dataset access).
insets	The amount of space to leave around the outside of the plot.
outline-stroke	The pen stroke used to draw an outline around the plot area.
outline-paint	The paint used to draw an outline around the plot area.
background-paint	The color used to draw the background of the plot area.
background-image	An image that is displayed in the background of the plot (optional).
background-alpha	The alpha transparency value used when coloring the plot's background, and also when drawing the background image (if there is one).
foreground-alpha	The alpha transparency used to draw items in the plot's foreground.
no-data-message	A string that is displayed by some plots when there is no data to display.
no-data-message-font	The font used to display the "no data" message.
data-area-ratio	The aspect ratio for the data area.

All subclasses will inherit these core attributes.

23.18.4 Usage

To customise the appearance of a plot, you first obtain a reference to the plot as follows:

```
Plot plot = myChart.getPlot();
```

With this reference, you can change the appearance of the plot by modifying it's attributes. For example:

```
plot.setBackgroundPaint(Color.yellow);
plot.setNoDataMessage("There is no data.");
```

Very often, you will find it necessary to cast the `Plot` object to a specific subclass so that you can access attributes that are defined by the subclass. Refer to the usage notes for each subclass for more details.

23.18.5 The Plot Background

The *background area* for a plot is the area inside the plot's axes (if the plot has axes)—it does not include the chart titles, the legend or the axis labels.

By default, the background area for most plot's in JFreeChart is white. You can easily change this using code similar to the following:

```
Plot plot = myChart.getPlot();
plot.setBackgroundPaint(Color.yellow);
```

You can also add an image to the background area. The image will be stretched to fill the plot area:

```
plot.setBackgroundImage(myImage);
```

Both the background paint and the background image can be drawn using an alpha-transparency, you can set this as follows:

```
plot.setBackgroundAlpha(0.6f);
```

There are similar methods in the `JFreeChart` class that allow you to control the background area for the chart (which encompasses the entire chart area).

23.18.6 Methods

The `JFreeChart` class expects every plot to implement the `draw(...)` method, and uses this to draw the plot in a specific area via a `Graphics2D` instance. You won't normally need to call this method yourself:

```
public abstract void draw(Graphics2D g2, Rectangle2D plotArea,
ChartRenderingInfo info);
```
Draws the chart using the supplied `Graphics2D`. The plot should be drawn within the `plotArea`.

If you wish to record details of the items drawn within the plot, you need to supply a `ChartRenderingInfo` object. Once the drawing is complete, this object will contain a lot of information about the plot. If you don't want this information, pass in `null`.

23.18.7 Notes

Refer to specific subclasses for information about setting the colors, shapes and line styles for data drawn by the plot.

23.19 PlotException

23.19.1 Overview

A general purpose exception that can be generated by subclasses of `Plot`.

23.19.2 Notes

At the current time, there isn't any code that throws this type of exception, but the class is being retained for future use.

23.20 PlotNotCompatibleException

23.20.1 Overview

An exception that indicates that an attempt has been made to assign a plot to a chart where the plot is not compatible with the chart's current `Dataset`. For example, an `XYPlot` will not work with a `CategoryDataset`.

23.20.2 Constructors

To create a new exception:

```
public PlotNotCompatibleException(String message);
```
Creates a new exception.

23.20.3 Notes

`PlotNotCompatibleException` class is a subclass of `RuntimeException`.

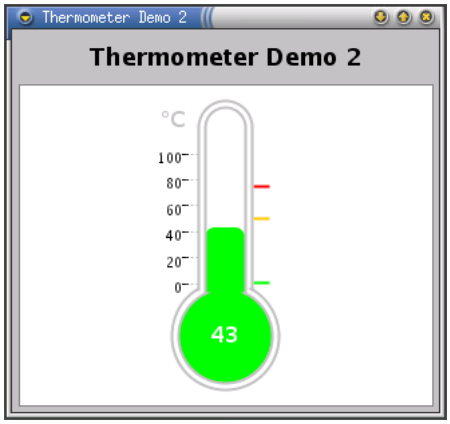
See Also

`AxisNotCompatibleException`.

23.21 ThermometerPlot

23.21.1 Overview

A plot that displays a single value in a thermometer-style representation.



You can define three sub-ranges on the thermometer scale to provide some context for the displayed value: the *normal*, *warning* and *critical* sub-ranges. The color of the "mercury" in the thermometer can be configured to change for each sub-range.

By default, the display range for the thermometer is fixed (using the overall range specified by the user). However, there is an option to automatically adjust the thermometer scale to display only the sub-range in which the current value falls. This allows the current data value to be displayed with more precision.

23.21.2 Constructors

To create a new `ThermometerPlot`:

```
public ThermometerPlot(MeterDataset data);
```
Creates a thermometer with default settings, using the supplied dataset.

If you want to have more control over the appearance of the thermometer:

`public ThermometerPlot(MeterDataset data, Insets insets,`
`Paint backgroundPaint, Image backgroundImage, float backgroundAlpha,`
`Stroke outlineStroke, Paint outlinePaint, float foregroundAlpha);`
Creates a thermometer with the supplied settings and dataset.

23.21.3 Methods

The current value can be displayed as text in the thermometer bulb or to the right of the thermometer. To set the position:

`public void setValueLocation(int location);`
Sets the position of the value label. Use one of the constants: `NONE`, `RIGHT` or `BULB`.

The font for the value label can be set as follows:

`public void setValueFont(Font font);`
Sets the font used to display the current value.

Similarly, the paint for the value label can be set as follows:

`public void setValuePaint(Paint paint);`
Sets the paint used to display the current value.

You can set a formatter for the value label:

`public void setValueFormatter(NumberFormat formatter);`
Sets the formatter for the value label.

To set the overall range of values to be displayed in the thermometer:

`public void setRange(double lower, double upper);`
Sets the lower and upper bounds for the value that can be displayed in the thermometer. If the data value is outside this range, the thermometer will be drawn as "empty" or "full".

You can specify the bounds for any of the three sub-ranges:

`public void setSubrange(int subrange, double lower, double upper);`
Sets the lower and upper bounds for a sub-range. Use one of the constants `NORMAL`, `WARNING` or `CRITICAL` to indicate the sub-range.

In addition to the actual bounds for the sub-ranges, you can specify *display bounds* for each sub-range:

`public void setDisplayBounds(int range, double lower, double upper);`
Sets the lower and upper bounds of the display range for a sub-range. The display range is usually equal to or slightly bigger than the actual bounds of the sub-range.

The display bounds are only used if the thermometer axis range is automatically adjusted to display the current sub-range. You can set a flag that controls whether or not this automatic adjustment happens:

`public void setFollowDataInSubranges(boolean flag);`
If `true`, the thermometer range is adjusted to display only the current sub-range (which displays the value with greater precision). If `false`, the overall range is displayed at all times.

By default, this flag is set to `false`.

To set the default color of the "mercury" in the thermometer:

`public void setMercuryPaint(Paint paint);`
Sets the default color of the mercury in the thermometer.

To set the color of the mercury for each sub-range:

`public void setSubrangePaint(int range, Paint paint);`
Sets the paint used for the mercury when the data value is within the specified sub-range. Use one of the constants `NORMAL`, `WARNING` or `CRITICAL` to indicate the sub-range.

The sub-range mercury colors are only used if the `UseSubrangePaint` flag is set to `true` (the default):

`public void setUseSubrangePaint(boolean flag);`
Sets the flag that controls whether or not the sub-range colors are used for the mercury in the thermometer.

To show grid lines within the thermometer stem:

`public void setShowValueLines(boolean flag);`
Sets a flag that controls whether or not grid lines are displayed inside the thermometer stem.

To control the color of the thermometer outline:

`public void setThermometerPaint(Paint paint);`
Sets the paint used to draw the outline of the thermometer.

To control the pen used to draw the thermometer outline:

`public void setThermometerStroke(Stroke stroke);`
Sets the stroke used to draw the outline of the thermometer.

You can control the amount of white space at the top and bottom of the thermometer:

`public void setPadding(Spacer padding);`
Sets the padding around the thermometer. This is controlled using a `Spacer` object.

23.21.4 Notes

The `ThermometerPlot` class was originally contributed by Bryan Scott from the Australian Antarctic Division.

The `JThermometer` class provides a simple (but incomplete) Javabean wrapper for this class.

The `MeterDataset` class can return information about the "normal", "warning" and "critical" ranges, but this information is ignored by this class. The ranges are, instead, defined using attributes in the `ThermometerPlot` class.

Various dimensions for the thermometer (for example, the bulb radius) are hard-coded constants in the current implementation. A useful enhancement would be to replace these constants with attributes that could be modified via methods in the `ThermometerPlot` class.

The `ThermometerDemo` class in the `com.jrefinery.chart.demo` package provides a working example of this class.

See Also

`MeterDataset`.

23.22 VerticalCategoryPlot

23.22.1 Overview

This plot draws a chart using data from a `CategoryDataset`, where the categories are plotted against the horizontal axis and the numerical data is plotted against the vertical axis.

23.22.2 Constructors

The simplest constructor requires only the axes to be specified:

> `public VerticalCategoryPlot(CategoryAxis domainAxis, ValueAxis rangeAxis);`
> Creates a vertical category plot. Default values are assumed for most attributes.

See Also

`HorizontalCategoryPlot`.

23.23 VerticalValuePlot

23.23.1 Overview

An interface that returns data range in the "vertical direction" for a two-dimensional plot. The values could be from the dataset domain or range, depending on the orientation of the plot.

23.23.2 Methods

To get the range:

`public Range getVerticalDataRange(ValueAxis axis);`
Returns the range of data values in the vertical direction for the plot.

To get the vertical axis:

`public ValueAxis getVerticalValueAxis();`
Returns the vertical value axis.

23.23.3 Notes

This interface is known to be implemented by `VerticalCategoryPlot` and `XYPlot`.

See Also
`HorizontalValuePlot`.

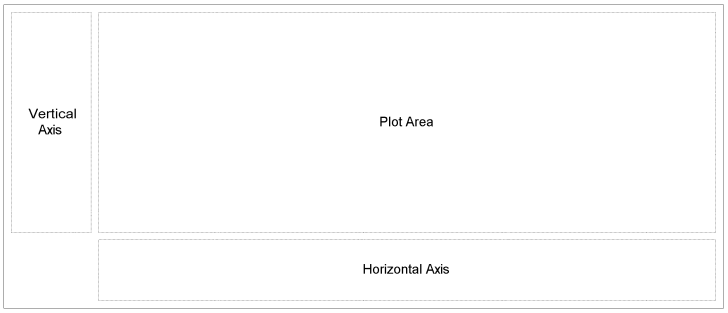
23.24 XYPlot

23.24.1 Overview

Draws a visual representation of data from an `XYDataset`, where the domain axis (or horizontal axis) measures the x-values and the range axis (or vertical axis) measures the y-values.

23.24.2 Layout

Axes are laid out at the left and bottom of the drawing area. The space allocated for the axes is determined automatically. The following diagram shows how this area is divided:



Determining the dimensions of these regions is an awkward problem. The plot area can be resized arbitrarily, but the vertical axis and horizontal axis sizes are more difficult. Note that the height of the vertical axis is related to the height of the horizontal axis, and, likewise, the width of the vertical axis is related to the width of the horizontal axis. This results in a "chicken and egg" problem, because changing the width of an axis can affect its height (especially if the tick units change with the resize) and changing its height can affect the width (for the same reason).

23.24.3 Renderers

The `XYPlot` class delegates drawing of individual data items to an `XYItemRenderer`. A number of renderer implementations are available:

- `StandardXYItemRenderer`;
- `HighLowRenderer`;
- `CandlestickRenderer`;
- `AreaXYRenderer`;

23.24.4 Gridlines

By default, the plot will draw *gridlines* in the background of the plot area. Vertical lines are drawn for each tick mark on the domain axis, and horizontal lines are drawn for each tick mark on the range axis.

You can customise both the color (`Paint`) and line-style (`Stroke`) of the grid-lines. For example, to change the grid lines to solid black lines:

```
XYPlot plot = myChart.getXYPlot();
plot.setDomainGridStroke(new BasicStroke(0.5f));
plot.setDomainGridPaint(Color.black);
plot.setRangeGridStroke(new BasicStroke(0.5f));
plot.setRangeGridPaint(Color.black);
```

If you prefer to have no gridlines at all, you can turn them off:

```
XYPlot plot = myChart.getXYPlot();
plot.setDomainGridVisible(false);
plot.setRangeGridVisible(false);
```

Note that the settings for the domain grid lines and the range grid lines are independent of one another.

23.24.5 Annotations

You can add annotations to a chart to highlight particular data items. For example, to add the text "Hello World!" to a plot:

```
XYPlot plot = myChart.getXYPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

To clear all annotations:

```
plot.clearAnnotations();
```

23.24.6 Constructors

The simplist constructor requires just the dataset and axes to be specified:

```
public XYPlot(XYDataset data,
ValueAxis domainAxis, ValueAxis rangeAxis);
```

Creates an XY plot using a `StandardXYItemRenderer`. Default attributes are used where necessary.

To create a plot with a specific renderer:

```
public XYPlot(XYDataset data,
ValueAxis domainAxis, ValueAxis rangeAxis, XYItemRenderer renderer);
```
Creates an XY plot with a specific renderer.

23.24.7 Methods

To get the current renderer for the plot:

```
public XYItemRenderer getRenderer();
```
Returns the current renderer.

To set a new renderer for the plot:

```
public void setRenderer(XYItemRenderer renderer);
```
Sets a new renderer.

You can add one or more "markers" to a plot to indicate important values in the domain or range. A marker is a constant value, represented using the `Marker` class.

To add a marker along the domain axis:

```
public void addDomainMarker(Marker marker);
```
Adds a marker for the domain axis. This is usually represented as a vertical line on the plot.

To add a marker along the range axis:

```
public void addRangeMarker(Marker marker);
```
Adds a marker for the range axis. This is usually represented as a horizontal line on the plot.

To clear all domain markers:

```
public void clearDomainMarkers();
```
Clears all the domain markers.

Likewise, to clear all range markers:

```
public void clearRangeMarkers();
```
Clears all the range markers.

23.24.8 Notes

`XYPlot` implements both `HorizontalValuePlot` and `VerticalValuePlot`, enabling the axes to automatically determine the range of data that is available for the plot.

It is possible to display time series data with `XYPlot` by employing a `HorizontalDateAxis` in place of the usual `HorizontalNumberAxis`. In this case, the x-values are interpreted as milliseconds as used in `java.util.Date`.

See Also

`Plot`, `OverlaidXYPlot`, `XYItemRenderer`.

24 Package: org.jfree.chart.renderer

24.1 Overview

This package contains interfaces and classes that are used to implement renderers, plug-in objects that are responsible for drawing data inside the plot area on a chart.

24.2 AbstractCategoryItemRenderer

24.2.1 Overview

A base class that can be used to implement a new *category item renderer* (a class that implements the `CategoryItemRenderer` interface).

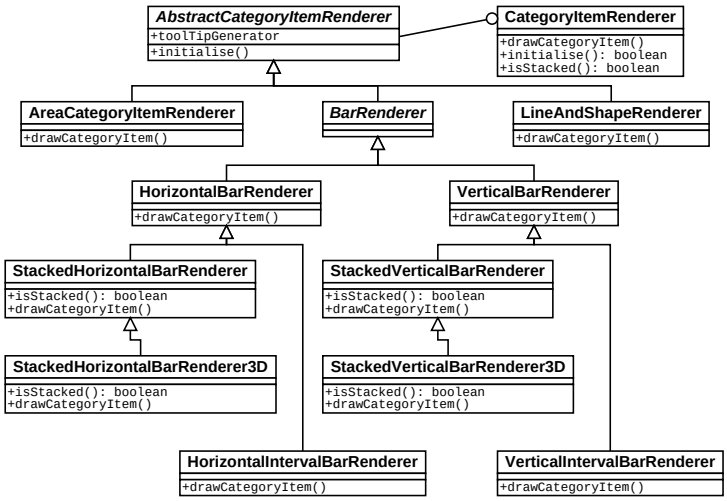


Figure 8: Category item renderers

24.2.2 Constructors

All of the constructors for this class are `protected`, since this is an `abstract` class—you never create an instance directly, you must use a subclass.

The default constructor creates a renderer with no tooltip generator and no URL generator. The other constructors allow you to supply a tooltip generator (some instance of `CategoryToolTipGenerator`) and/or a URL generator (some instance of `CategoryURLGenerator`).

24.2.3 Methods

The following method is called once every time the chart is drawn:

> `public void initialise(...);`
> Performs any initialisation required by the renderer. The default implementation simply stores a local reference to the `info` object (which may be `null`).

The number of rows and columns in the dataset (a `TableDataset`) is cached by the renderer in the `initialise(...)` method.

To get the row count:

```
public int getRowCount();
```
Returns the row count.

To get the column count:

```
public int getColumnCount();
```
Returns the column count.

24.2.4 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

See Also

`CategoryItemRenderer`.

24.3 AbstractRenderer

24.3.1 Overview

An abstract class that implements the `Renderer` interface. This class provides methods for controlling the paint, stroke and shape objects used by most renderers.

This base class is extended by both the `AbstractCategoryItemRenderer` class and the `AbstractXYItemRenderer` class.

24.3.2 Attributes

The attributes maintained by the `AbstractRenderer` class are listed in Table 9.

24.3.3 Setting Series Colors

Renderers are responsible for drawing the data items within a plot, so this class provides attributes for controlling the colors that will be used.

Colors are defined on a "per series" basis, and stored in a lookup table (an instance of `PaintTable`). The table has two rows—colors from the first row are used for the primary dataset, and colors from the second row are used for the secondary dataset. The renderer may be responsible for drawing items from only one dataset, in which case one of the rows in the paint table will remain unused.

There is a default mechanism to automatically populate the paint table with default colors. However, you can manually update the paint table at any time. First, you need to obtain a reference to the renderer:

```
CategoryPlot plot = myChart.getCategoryPlot();
AbstractRenderer r1 = (AbstractRenderer) plot.getRenderer();
AbstractRenderer r2 = (AbstractRenderer) plot.getSecondaryRenderer();
```

Attribute:	Description:
plot	The `Plot` that the renderer is assigned to.
supplier	A `DrawingSupplier`, provides a never-ending (although possibly repeating) sequence of `Paint`, `Stroke` and `Shape` objects. The renderer will use these, if necessary, to populate its internal lookup tables.
default-paint	The default paint, returned if the paint table is not active.
paint-table-active	A flag that controls whether the renderer uses the *paint-table* or the *default-paint* to determine the color of the data items (in most cases, the paint table will be used).
paint-table	A lookup table for the colors used for each data item.
default-outline-paint	The default outline paint, returned if the outline paint table is not active.
outline-paint-table-active	A flag that controls whether the renderer uses the *outline-paint-table* or the *default-outline-paint* to determine the color of the outline of the data items.
outline-paint-table	A lookup table for the colors used to outline each data item.
default-stroke	The default stroke, returned if the stroke table is not active.
stroke-table-active	A flag that controls whether the renderer uses the *stroke-table* or the *default-stroke* to determine the stroke used for each data item (in most cases, the default stroke will be used).
stroke-table	A lookup table for the strokes used for each data item.
default-outline-stroke	The default outline stroke, returned if the outline stroke table is not active.
outline-stroke-table-active	A flag that controls whether the renderer uses the *outline-stroke-table* or the *default-outline-stroke* to determine the stroke used for each data item (in most cases, the default outline stroke will be used).
outline-stroke-table	A lookup table for the outline stroke used for each data item.

Table 9: Attributes for the `AbstractRenderer` class

The code is similar for charts that use `XYPlot`:

```
XYPlot plot = myChart.getXYPlot();
AbstractRenderer r1 = (AbstractRenderer) plot.getRenderer();
AbstractRenderer r2 = (AbstractRenderer) plot.getSecondaryRenderer();
```

Note that many charts do not use a secondary renderer.

To update the paint table:

```
// the following methods set the primary dataset colors
// by default...
r1.setSeriesPaint(0, Color.red);
r1.setSeriesPaint(1, Color.green);
r1.setSeriesPaint(2, Color.blue);
```

To set the colors for the secondary dataset (if one is being used), you need to use the `setSeriesPaint(...)` method that incorporates the dataset index:

```
// primary dataset index = 0
r1.setSeriesPaint(0, 0, Color.red);
r1.setSeriesPaint(0, 1, Color.green);
r1.setSeriesPaint(0, 2, Color.blue);

// secondary dataset index = 1
r1.setSeriesPaint(1, 0, Color.orange);
r1.setSeriesPaint(1, 1, Color.yellow);
r1.setSeriesPaint(1, 2, Color.gray);
```

24.3.4 Setting Series Shapes

Renderers are initialised so that a range of default shapes are available if required. These are stored in a lookup table that is initially empty. The lookup table has two rows (one for the primary dataset, and one for the secondary dataset), and can have any number of columns (one per series). When the renderer requires a `Shape`, it uses the dataset index (primary or secondary) and the series index to read a shape from the lookup table. If the value is `null`, then the renderer turns to the `DrawingSupplier` for a new shape—the next shape is returned by the `getNextShape()` method.

If you require more control over the shapes that are used for your plots, you can populate the lookup table yourself using the `setSeriesShape(...)` method. The shape you supply can be any instance of `Shape`, but should be centered on *(0, 0)* in Java2D space (so that JFreeChart can position the shape at any data point).

Here is some sample code that sets four custom shapes for the primary dataset in an `XYPlot`:

```
XYPlot plot = chart.getXYPlot();
XYItemRenderer r = plot.getRenderer();
if (r instanceof StandardXYItemRenderer) {
    StandardXYItemRenderer renderer = (StandardXYItemRenderer) r;
    renderer.setPlotShapes(true);
    renderer.setDefaultShapeFilled(true);
    renderer.setSeriesShape(0, new Ellipse2D.Double(-3.0, -3.0, 6.0, 6.0));
    renderer.setSeriesShape(1, new Rectangle2D.Double(-3.0, -3.0, 6.0, 6.0));
    GeneralPath s2 = new GeneralPath();
    s2.moveTo(0.0f, -3.0f);
    s2.lineTo(3.0f, 3.0f);
    s2.lineTo(-3.0f, 3.0f);
    s2.closePath();
    renderer.setSeriesShape(2, s2);
    GeneralPath s3 = new GeneralPath();
    s3.moveTo(-1.0f, -3.0f);
    s3.lineTo(1.0f, -3.0f);
    s3.lineTo(1.0f, -1.0f);
    s3.lineTo(3.0f, -1.0f);
    s3.lineTo(3.0f, 1.0f);
    s3.lineTo(1.0f, 1.0f);
    s3.lineTo(1.0f, 3.0f);
    s3.lineTo(-1.0f, 3.0f);
    s3.lineTo(-1.0f, 1.0f);
    s3.lineTo(-3.0f, 1.0f);
    s3.lineTo(-3.0f, -1.0f);
    s3.lineTo(-1.0f, -1.0f);
    s3.closePath();
    renderer.setSeriesShape(3, s3);
}
```

24.4 AbstractXYItemRenderer

24.4.1 Overview

A convenient base class for creating new `XYItemRenderer` implementations.

24.4.2 Constructors

This class provides a default constructor which allocates storage for the list of property change listeners.

24.4.3 Methods

To register a `PropertyChangeListener` with the renderer:

> `public void addPropertyChangeListener(PropertyChangeListener listener);`
> Registers a listener so that it receives notification of any changes to the renderer.

If an object no longer wishes to receive property change notifications:

> `public void removePropertyChangeListener(PropertyChangeListener listener);`
> Removes a listener so that it no longer receives notification of changes to the renderer.

A default method is supplied for displaying a *domain marker* as a vertical line on the plot:

> `public void drawDomainMarker(...);`
> Draws a vertical line to represent a "marker" on the domain axis.

A default method is supplied for displaying a *range marker* as a horizontal line on the plot:

> `public void drawRangeMarker(...);`
> Draws a horizontal line to represent a "marker" on the range axis.

24.4.4 Notes

This class provides a property change mechanism to support the requirements of the `XYItemRenderer` interface.

The methods for drawing domain and range markers can be overridden by subclasses, if necessary.

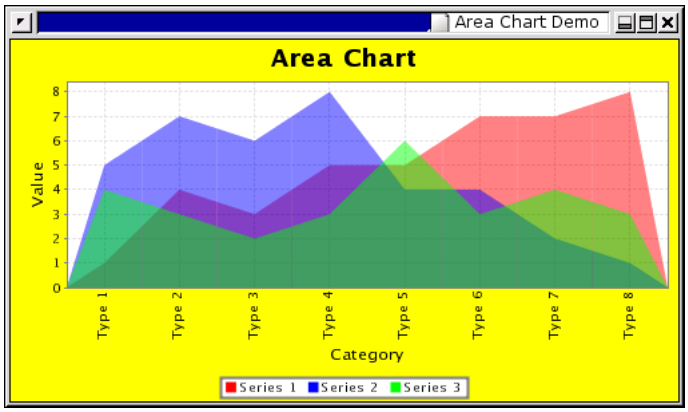
See Also

`XYItemRenderer`, `XYPlot`.

24.5 AreaRenderer

24.5.1 Overview

An *area renderer* draws each item in a `CategoryDataset` using a polygon that fills the area between the x-axis and the data point.



This class is designed for use with the `VerticalCategoryPlot` class.

24.5.2 Notes

The `createAreaChart(...)` method in the `ChartFactory` class will create a default chart that uses this renderer.

This class extends `AbstractCategoryItemRenderer`.

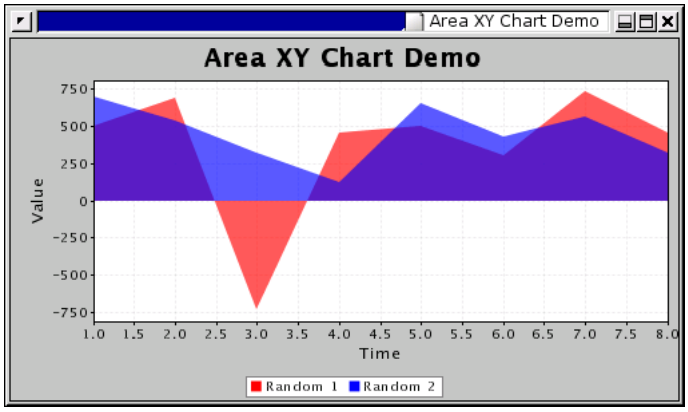
See Also

`AreaXYRenderer`.

24.6 AreaXYRenderer

24.6.1 Overview

An *area XY renderer* draws each item in an `XYDataset` using a polygon that fills the area between the x-axis and the data point:



This renderer is designed to be used with the `XYPlot` class.

24.6.2 Constructors

The default constructor sets up the renderer to draw area charts:

```
public AreaXYItemRenderer();
```
Creates a new renderer.

You can change the appearance of the chart by specifying the type:

```
public AreaXYItemRenderer(int type);
```
Creates a new `AreaXYItemRenderer` using one of the following types: `SHAPES`, `LINES`, `SHAPES_AND_LINES`, `AREA`, `AREA_AND_SHAPES`.

24.6.3 Notes

This class extends `AbstractXYItemRenderer`.

You can see from this second constructor that this class uses code copied from the `StandardXYItemRenderer` class, and that some additional work is required to eliminate the duplication. One option (still under consideration) for a future version of JFreeChart is to merge the two classes.

See Also

`AreaRenderer`.

24.7 BarRenderer

24.7.1 Overview

A base class that is used to implement various *category item renderers* that represent data using bars. Subclasses include:

- `HorizontalBarRenderer`;
- `VerticalBarRenderer`;

24.7.2 Methods

To control the amount of space between bars *within a category*:

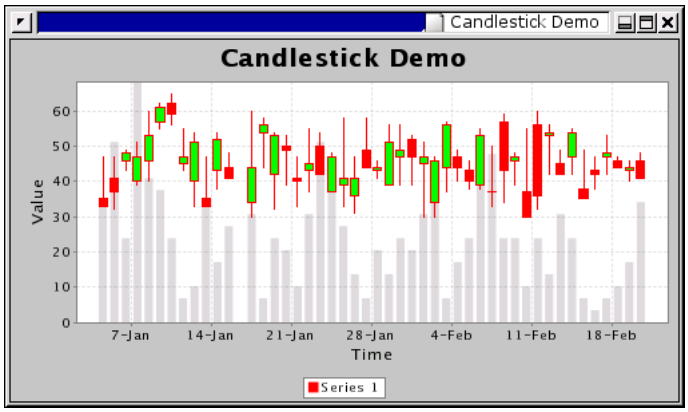
```
public setItemMargin(double percent);
```
Sets the amount of space (as a percentage of the overall space available for drawing all the bars) to be allocated to the gaps between bars that are in the same category.

The amount of space between categories is controlled by the `CategoryAxis`.

24.8 CandlestickRenderer

24.8.1 Overview

A *candlestick renderer* draws each item from a `HighLowDataset` as a box with lines extending from the top and bottom. Candlestick charts are typically used to display financial data—the box represents the open and closing prices, while the lines indicate the high and low prices for a trading period (often one day).



This renderer is designed for use with the XYPlot class.

This renderer also has the ability to represent volume information in the background of the chart.

24.8.2 Constructors

To create a new renderer:

`public CandlestickRenderer(double candleWidth);`
Creates a new renderer.

24.8.3 Methods

To set the width of the candles (in points):

`public void setCandleWidth(double width);`
Sets the width of each candle. If the value is negative, then the renderer will automatically determine a width each time the chart is redrawn.

To set the color used to fill candles when the closing price is higher than the opening price (the price has moved up):

`public void setUpPaint(Paint paint);`
Sets the fill color for candles where the closing price is higher than the opening price.

To set the color used to fill candles when the closing price is lower than the opening price (the price has moved down):

`public void setDownPaint(Paint paint);`
Sets the fill color for candles where the closing price is lower than the opening price.

To control whether or not volume bars are drawn in the background of the chart:

`public void setDrawVolume(boolean flag);`
Controls whether or not volume bars are drawn in the background of the chart.

These methods will fire a property change event that will be picked up by the XYPlot class, triggering a chart redraw.

24.8.4 Notes

This renderer requires a `HighLowDataset`.

24.9 CategoryItemRenderer

24.9.1 Overview

The interface that must be supported by a *category item renderer*. A renderer is a plug-in for the `CategoryPlot` class that is responsible for drawing individual data items.

A number of different renderers have been developed, allowing different chart types to be generated easily. The following table lists the renderers that have been implemented to date:

Class:	Description:
`HorizontalBarRenderer`	Represents data using horizontal bars (anchored at zero).
`VerticalBarRenderer`	Represents data using vertical bars (anchored at zero).
`HorizontalIntervalBarRenderer`	Draws intervals using horizontal bars. This renderer can be used to create simple Gantt charts.
`LineAndShapeRenderer`	Draws lines and/or shapes to represent data.
`AreaCategoryItemRenderer`	Used to create area charts.
`StackedHorizontalBarRenderer`	Used to create a horizontal stacked bar chart.
`StackedVerticalBarRenderer`	Used to create a vertical stacked bar chart.

24.9.2 Methods

The interface defines an initialisation method:

`public void initialise(...);`
This method is called at the start of every chart redraw. It gives the renderer a chance to precalculate any information it might require later when rendering individual data items.

For data range calculations, the `CategoryPlot` class needs to know whether or not the renderer stacks values. This can be determined via the following method:

`public boolean isStacked();`
Returns `true` if the values are stacked, and `false` otherwise.

The most important method is the one that actually draws a data item:

`public Shape drawCategoryItem(...);`
Draws one item on a category plot.

24.9.3 Notes

Classes that implement the `CategoryItemRenderer` interface are used by the `CategoryPlot` class. They cannot be used by the `XYPlot` class (which uses implementations of the `XYItemRenderer` interface).

See Also

CategoryPlot, AbstractCategoryItemRenderer.

24.10 ClusteredXYBarRenderer

24.10.1 Overview

Not yet documented.

24.11 DefaultDrawingSupplier

24.11.1 Overview

A default implementation of the DrawingSupplier interface.

24.11.2 Constructors

In addition to the default constructor, you can use the following constructor to create a new supplier:

```
public DefaultDrawingSupplier(Paint[] paintSequence,
Paint[] outlinePaintSequence, Stroke[] strokeSequence,
Stroke[] outlineStrokeSequence, Shape[] shapeSequence);
```

Creates a new supplier using the supplied arrays as the source for each of the sequences.

24.11.3 Methods

This class implements all the methods in the DrawingSupplier interface.

24.11.4 Notes

All renderers are automatically assigned their own default supplier. For a chart that uses multiple renderers (a combined chart, for example), you will usually want to create a single supplier to be shared among all renderers. This ensures that the colors, line styles and shapes for the entire chart come from a single source.

24.12 DrawingSupplier

24.12.1 Overview

A *drawing supplier* provides a never-ending (although possibly repeating) sequence of Paint, Stroke, and Shape objects that renderers can use to populate their internal lookup tables. By using a single supplier for multiple renderers (in a combined chart, for example), you can ensure that all series are allocated a unique color, stroke and/or shape.

24.12.2 Methods

To get the next *paint* in a sequence maintained by the supplier:

`public Paint getNextPaint();`
Returns the next paint object in the sequence.

To get the next *outline paint* in a sequence maintained by the supplier:

`public Paint getNextOutlinePaint();`
Returns the next outline paint in the sequence.

To get the next *stroke* in a sequence maintained by the supplier:

`public Stroke getNextStroke();`
Returns the next stroke in the sequence.

To get the next *outline stroke* in a sequence maintained by the supplier:

`public Stroke getNextOutlineStroke();`
Returns the next outline stroke in the sequence.

To get the next *shape* in a sequence maintained by the supplier:

`public Shape getNextShape();`
Returns the next shape in the sequence.

24.12.3 Notes

The `DefaultDrawingSupplier` class provides a default implementation of this interface, but you can also provide your own implementation if you want to.

The `OverlaidXYPlotDemo` application illustrates the use of this interface to coordinate the renderers in an overlaid plot.

See Also
`Renderer`.

24.13 HighLow

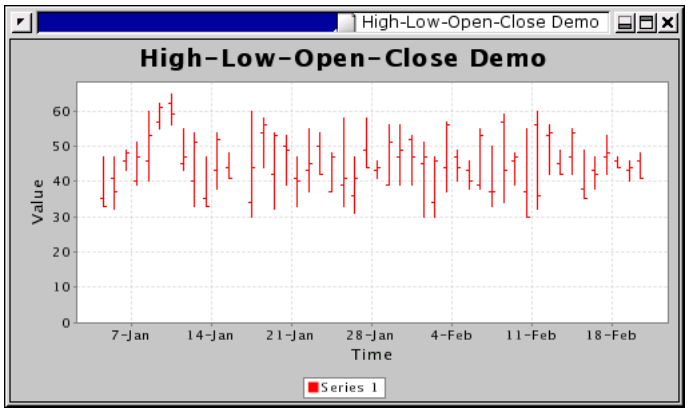
24.13.1 Overview

Represents one item used by a `HighLowRenderer` during the rendering process.

24.14 HighLowRenderer

24.14.1 Overview

A *high-low renderer* draws each item in an `XYDataset` using lines to mark the "high-low" range for a trading period, plus small marks to indicate the "open" and "close" values.



This renderer is designed for use with the `XYPlot` class. It requires a `HighLowDataset`.

24.14.2 Constructors

To create a new renderer:

`public HighLowRenderer(XYToolTipGenerator toolTipGenerator);`
Creates a new renderer with the supplied tool tip generator (`null` permitted).

24.14.3 Methods

Implements the `drawItem(...)` method defined in the `XYItemRenderer` interface.

24.14.4 Notes

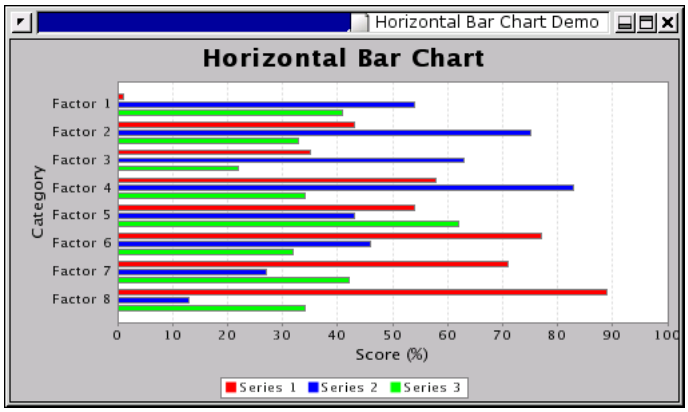
This renderer requires the dataset to be an instance of `HighLowDataset`.

The `createHighLowChart(...)` method in the `ChartFactory` class makes use of this renderer.

24.15 HorizontalBarRenderer

24.15.1 Overview

A *horizontal bar renderer* draws each item in a `CategoryDataset` as a horizontal bar.



This renderer is designed for use with the `HorizontalCategoryPlot` class.

24.15.2 Constructors

To create a new renderer:

> `public HorizontalBarRenderer(CategoryToolTipGenerator toolTipGenerator, CategoryURLGenerator urlGenerator);`
> Creates a new renderer with the specified tool tip and URL generators (either or both may be `null`).

24.15.3 Methods

This class implements the methods in the `CategoryItemRenderer` interface.

24.15.4 Notes

The `ChartFactory` class uses this renderer to create horizontal bar charts (see the `createHorizontalBarChart(...)` method).

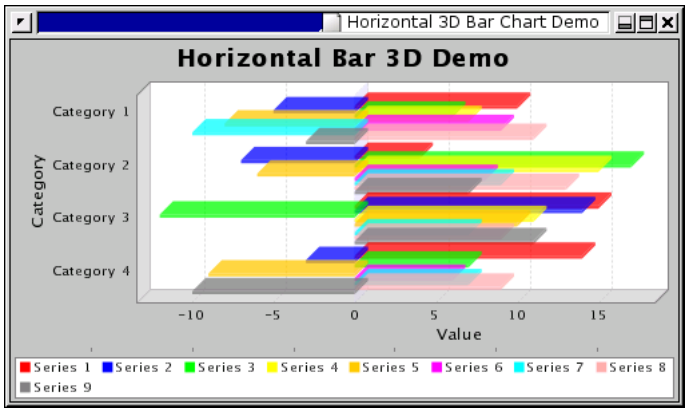
See Also

`StackedHorizontalBarRenderer`, `VerticalBarRenderer`.

24.16 HorizontalBarRenderer3D

24.16.1 Overview

A horizontal bar renderer that displays bars with a 3D effect.



This class extends the `HorizontalBarRenderer` class and implements the `Effect3D` interface.

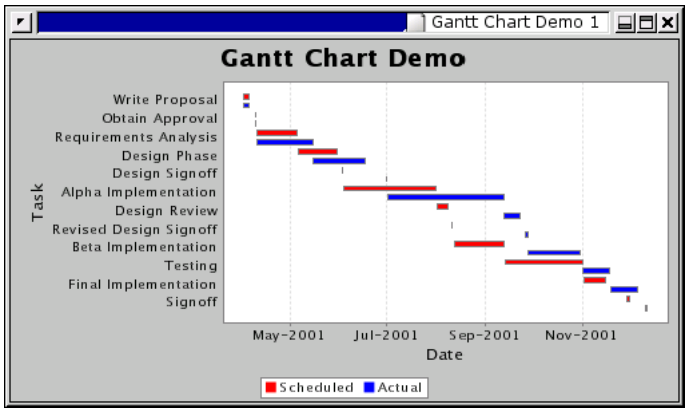
24.16.2 Notes

The `HorizontalBarChart3DDemo` application (included in the JFreeChart distribution) provides a demonstration of this renderer.

24.17 HorizontalIntervalBarRenderer

24.17.1 Overview

A *horizontal interval bar renderer* draws items in an `IntervalCategoryDataset` as horizontal bars.



This renderer is designed to work with the `HorizontalCategoryPlot` class.

24.17.2 Notes

This renderer is used to create simple Gantt charts (see the `GanttDemo` application for an example).

The `IntervalCategoryToolTipGenerator` can be used to generate tool tips with this renderer.

See Also

HorizontalCategoryPlot, CategoryItemRenderer.

24.18 HorizontalShapeRenderer

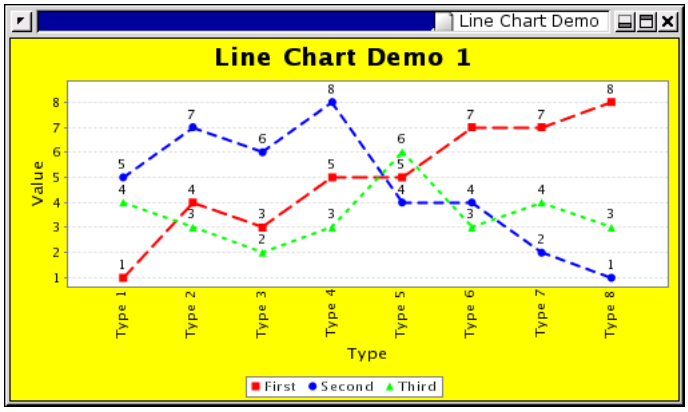
24.18.1 Overview

A *horizontal shape renderer...*

24.19 LineAndShapeRenderer

24.19.1 Overview

A *line and shape renderer* displays items in a `CategoryDataset` by drawing a shape at each data point, or connecting data points with straight lines, or both.



This renderer is designed for use with the `VerticalCategoryPlot` class.

24.19.2 Constructors

The default constructor creates a renderer that draws both shapes and lines:

> `public LineAndShapeRenderer();`
> Creates a new renderer that draws both shapes and lines.

The other constructor allows you to specify the type of renderer:

> `public LineAndShapeRenderer(int type);`
> Creates a new renderer of the specified type. Use one of the constants defined by this class: `SHAPES`, `LINES`, or `SHAPES_AND_LINES`.

24.19.3 Methods

This class implements the `drawCategoryItem(...)` method that is defined in the `CategoryItemRenderer` interface.

24.20 MinMaxCategoryRenderer

24.20.1 Overview

Not yet documented.

24.21 PaintTable

24.21.1 Overview

Not yet documented.

24.22 Renderer

24.22.1 Overview

A *renderer* is a plug-in object that is used to draw individual data items in a chart. This base interface defines methods that are shared by the more specific `CategoryItemRenderer` and `XYItemRenderer` interfaces.

24.22.2 Attributes

This interface defines accessor methods for a range of attributes that must be maintained by classes that implement this interface. The `AbstractRenderer` class provides a basic implementation.

See Also

`CategoryPlot`, `XYPlot`.

24.23 ReverseXYItemRenderer

24.23.1 Overview

Not yet documented.

24.24 ShapeTable

24.24.1 Overview

Not yet documented.

24.25 SignalRenderer

24.25.1 Overview

A plot that draws different signals depending on the direction of the data.

24.25.2 Notes

This was contributed by Sylvain Vieujot.

See Also

`Plot`.

24.26 StackedAreaRenderer

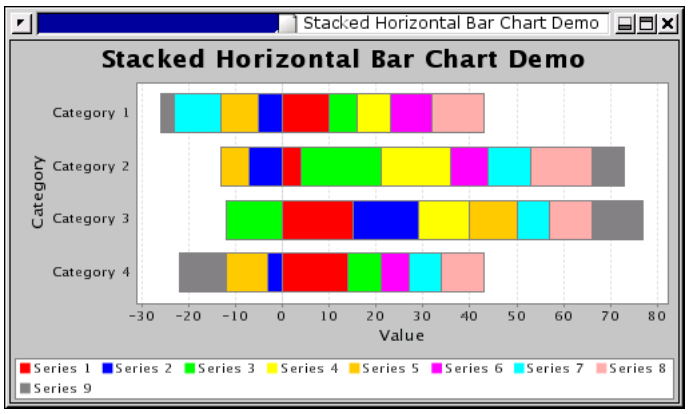
24.26.1 Overview

Not yet documented.

24.27 StackedHorizontalBarRenderer

24.27.1 Overview

A *stacked horizontal bar renderer* draws each item in a `CategoryDataset` in the form of "stacked" bars.



This renderer is designed for use with the `HorizontalCategoryPlot` class.

24.27.2 Methods

This class implements the methods in the `CategoryItemRenderer` interface.

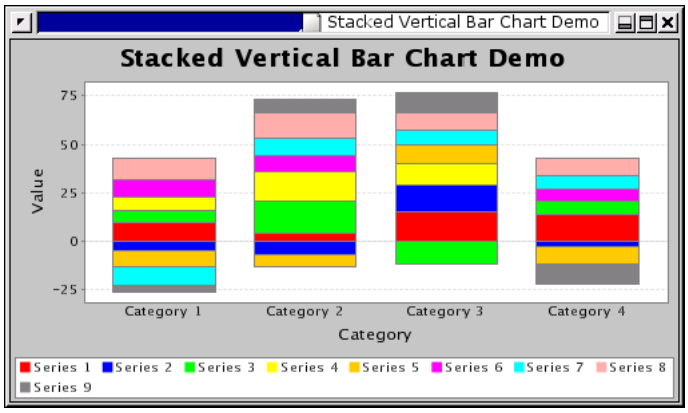
See Also

`StackedVerticalBarRenderer`.

24.28 StackedVerticalBarRenderer

24.28.1 Overview

A *stacked vertical bar renderer* draws each item in a `CategoryDataset` in the form of "stacked" bars.



This renderer is designed for use with the `VerticalCategoryPlot` class.

24.28.2 Methods

This class implements the methods in the `CategoryItemRenderer` interface.

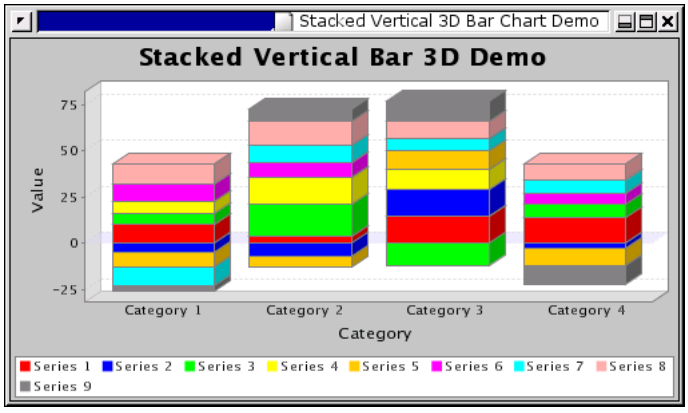
See Also

`StackedHorizontalBarRenderer`.

24.29 StackedVerticalBarRenderer3D

24.29.1 Overview

A *stacked vertical bar renderer (3D)* draws items from a `CategoryDataset` in the form of "stacked" bars with a 3D effect.



This renderer is designed for use with the `VerticalCategoryPlot` class.

24.29.2 Methods

This class implements the methods in the `CategoryItemRenderer` interface.

See Also

`StackedVerticalBarRenderer`.

24.30 StandardXYItemRenderer

24.30.1 Overview

A standard *renderer* for the `XYPlot` class. This renderer represents data by drawing lines between *(x, y)* data points. There is also a mechanism for drawing shapes or images at each at each *(x, y)* data point (with or without the lines).

24.30.2 Constructors

To create a `StandardXYItemRenderer`:

> `public StandardXYItemRenderer(int type);`
> Creates a new renderer. The `type` argument should be one of: `LINES`, `SHAPES` or `SHAPES_AND_LINES`.

24.30.3 Methods

To control whether or not the renderer draws lines between data points:

> `public void setPlotLines(boolean flag);`
> Sets the flag that controls whether or not lines are plotted between data points. The stroke and paint used for the lines is determined by the plot, per series.

To control whether or not the renderer draws shapes at each data point:

> `public void setPlotShapes(boolean flag);`
> Sets the flag that controls whether or not shapes are plotted at each data point.
>
> For each item, the shape to be plotted is obtained from the `getShape(...)` method which, unless overridden, delegates to the plot's `getShape(...)` method (which will return a different shape for each series).

When the renderer draws each shape, it can draw an outline of the shape, or it can fill the shape with a solid color. This is controlled by a protected method:

> `protected boolean isShapeFilled(...);`
> Returns a flag that controls whether or not the shape is filled.
>
> By default, this method returns the value from the `getDefaultShapeFilled()` method, but you can override the method in a subclass to customise the behaviour.

24.30.4 Notes

This class implements the `XYItemRenderer` interface.

The `XYPlot` class will use an instance of this class as its default renderer.

See Also

XYPlot, XYItemRenderer.

24.31 StrokeTable

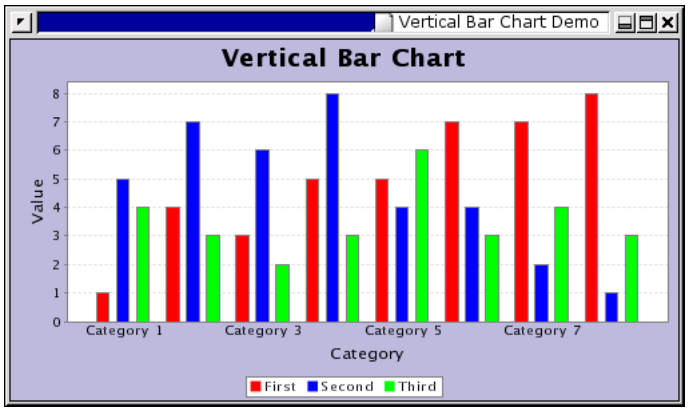
24.31.1 Overview

Not yet documented.

24.32 VerticalBarRenderer

24.32.1 Overview

A *vertical bar renderer* draws each data item in a `VerticalCategoryPlot` as an "upright" bar.



This class extends `BarRenderer` and implements the `CategoryItemRenderer` interface.

24.32.2 Constructors

To create a new renderer:

```
public VerticalBarRenderer(CategoryToolTipGenerator toolTipGenerator,
CategoryURLGenerator urlGenerator);
```

Creates a new renderer with the specified tool tip and URL generators (either or both may be `null`).

24.32.3 Methods

This class implements the methods in the `CategoryItemRenderer` interface.

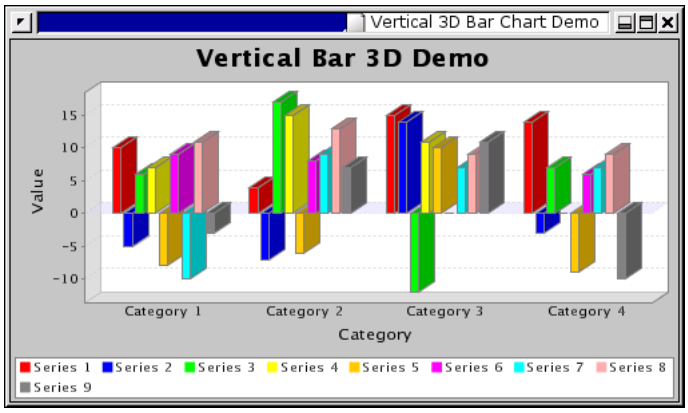
24.32.4 Notes

The `ChartFactory` class uses this renderer to create vertical bar charts (see the `createVerticalBarChart(...)` method).

24.33 VerticalBarRenderer3D

24.33.1 Overview

A renderer that draws items from a `CategoryDataset` using bars with a 3D effect.



This renderer is designed for use with the `VerticalCategoryPlot` class.

24.33.2 Notes

This class implements the `CategoryItemRenderer` interface.

24.34 VerticalIntervalBarRenderer

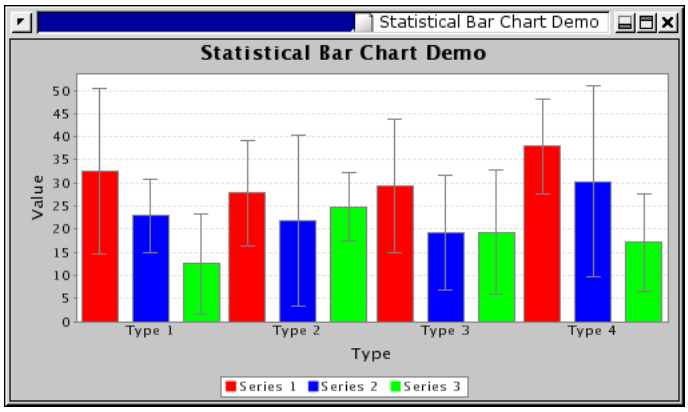
24.34.1 Overview

To be documented.

24.35 VerticalStatisticalBarRenderer

24.35.1 Overview

A *vertical statistical bar renderer* draws items from a `StatisticalCategoryDataset` in the form of bars with a line indicating the standard deviation.



This renderer is designed for use with the `VerticalCategoryPlot` class.

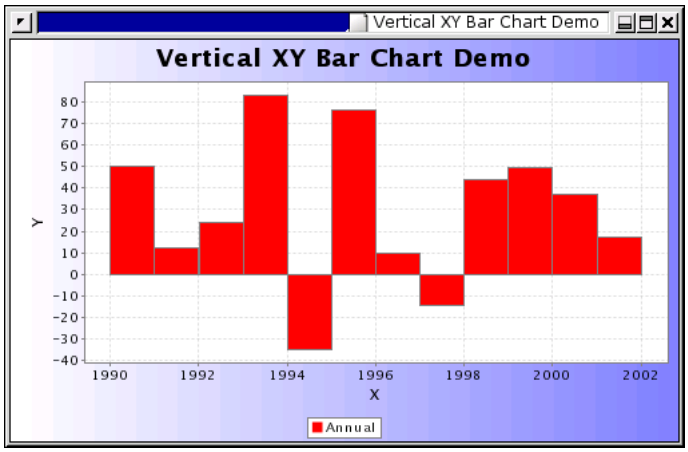
24.35.2 Notes

This class implements the `CategoryItemRenderer` interface.

24.36 VerticalXYBarRenderer

24.36.1 Overview

A *vertical XY bar renderer* draws items from an `IntervalXYDataset` in the form of vertical bars.



This renderer is designed to work with an `XYPlot`.

24.36.2 Constructors

The only constructor takes no arguments.

24.36.3 Methods

The `drawItem(...)` method handles the rendering of a single item for the plot.

24.36.4 Notes

This renderer casts the dataset to `IntervalXYDataset`, so you should ensure that the plot is supplied with the correct type of data. Refer to Javadoc HTML files and source code for further details.

See Also

`XYPlot`.

24.37 WindItemRenderer

24.37.1 Overview

A renderer that `XYPlot` uses to draw wind plots.

See Also

`XYPlot`.

24.38 XYBubbleRenderer

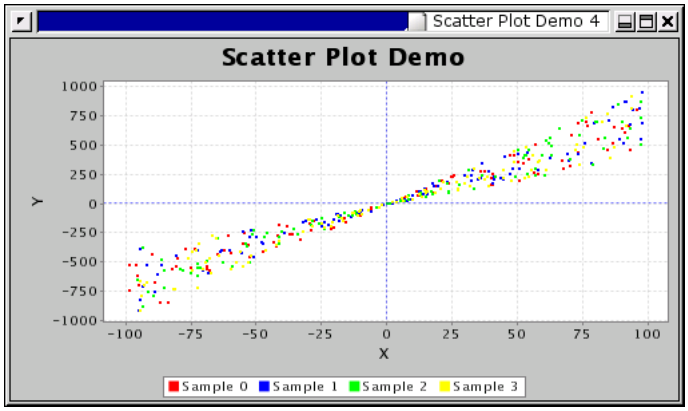
24.38.1 Overview

Not yet documented.

24.39 XYDotRenderer

24.39.1 Overview

An *XY dot renderer* displays items from an `XYDataset` by drawing a dot at each *(x, y)* point.



24.39.2 Notes

This class implements the `XYItemRenderer` interface and extends the `AbstractXYItemRenderer` class.

The `ScatterPlotDemo4` application (included in the JFreeChart distribution) provides a demonstration of this renderer.

24.40 XYItemRenderer

24.40.1 Overview

The interface that must be implemented by an *item renderer* so that it can work with an `XYPlot`. The item renderer is responsible for drawing the visual representation of each data item in a plot—by changing the renderer for a plot, you can change the appearance of the entire plot.

Figure 9 illustrates the hierarchy of classes that implement this interface.

The renderer provides a mechanism for generating tooltips (for charts displayed in a `ChartPanel`) and URLs for charts used in an HTML image map.

24.40.2 Methods

The `initialise` method is called once at the beginning of the chart drawing process, and gives the renderer a chance to initialise itself:

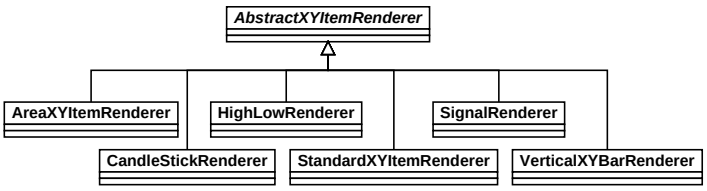


Figure 9: Renderer hierarchy

`public void initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset data, ChartRenderingInfo info);`
Initialises the renderer. If possible, a renderer will pre-calculate any values that help to improve the performance of the `drawItem(...)` method.

The `drawItem` method is responsible for drawing some representation of a particular data item within a plot:

```
public void drawItem(Graphics2D g2, Rectangle2D dataArea,
ChartRenderingInfo info, XYPlot plot,
ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset data, int series, int item, CrosshairInfo info);
```
Draws a single data item on behalf of `XYPlot`.

You can set your own *tooltip generator* and *URL generator* for the renderer.

24.40.3 Notes

Implementations of this interface include:

- `AreaXYRenderer`;
- `CandleStickRenderer`;
- `HighLowRenderer`;
- `StandardXYItemRenderer`;
- `SignalRenderer`;
- `VerticalXYBarRenderer`.

Some renderers require the a dataset that is a specific extension of `XYDataset`. For example, the `HighLowRenderer` requires a `HighLowDataset`.

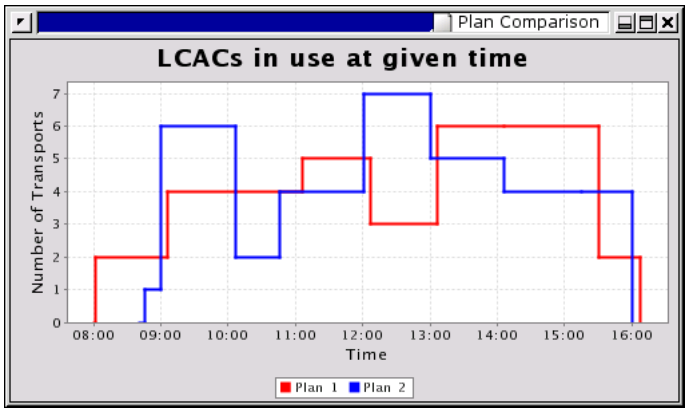
See Also

`AbstractXYItemRenderer`, `XYPlot`.

24.41 XYStepRenderer

24.41.1 Overview

An *XY step renderer* draws items from an `XYDataset` using "stepped" lines to connect each *(x, y)* point.



This renderer is designed for use with the `XYPlot` class.

24.41.2 Notes

The `XYStepChartDemo` class in the `com.jrefinery.chart.demo` package provides an example of this renderer in use.

24.42 YIntervalRenderer

24.42.1 Overview

Not yet documented.

25 Package: org.jfree.chart.tooltips

25.1 Introduction

This package contains interfaces and classes for generating tooltips. Section 11 contains information about using tool tips with JFreeChart.

25.2 CategoryToolTipGenerator

25.2.1 Overview

The interface that must be implemented by a *category tool tip generator*, a class that generates tool tips for each item in a `CategoryPlot`.

25.2.2 Methods

This interface defines a single method:

> `public String generateToolTip(CategoryDataset data, int series, Object category);`
> This method is called whenever the plot needs to generate a tool tip. It should return the tooltip text (which can be anything you want to make it).

25.2.3 Notes

The `StandardCategoryToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

To "install" a tool tip generator, use the `setToolTipGenerator(...)` method in the `AbstractCategoryItemRenderer` class.

See section 11 for information about using tool tips with JFreeChart.

See Also

`StandardCategoryToolTipGenerator`.

25.3 ContourToolTipGenerator

25.3.1 Overview

Not yet documented.

25.4 CustomXYToolTipGenerator

25.4.1 Overview

A tool tip generator (for use with an `XYItemRenderer`

25.4.2 Methods

To specify the text to use for the tool tips:

`public void addToolTipSeries(List toolTips);`
Adds the list of tool tips (for one series) to internal storage. These tool tips will be returned (without modification) by the generator for each data item.

25.4.3 Notes

See section 11 for information about using tool tips with JFreeChart.

See Also
`XYToolTipGenerator`.

25.5 HighLowToolTipGenerator

25.5.1 Overview

The interface that should be implemented by a *high-low tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a plot, and take full control over the tooltip text that is generated.

25.5.2 Methods

This interface defines a single method:

`public String generateToolTip(HighLowDataset data, int series, int item);`
This method is called whenever the plot needs to generate a tooltip. It should return the tooltip text (which can be anything you want to make it).

25.5.3 Notes

The `StandardHighLowToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

See section 11 for an overview of tool tips with JFreeChart.

See Also
`StandardHighLowToolTipGenerator`.

25.6 IntervalCategoryToolTipGenerator

25.6.1 Overview

Not yet documented.

25.7 PieToolTipGenerator

25.7.1 Overview

The interface that must be implemented by a *pie tool tip generator*, a class used to generate tool tip text for a pie chart.

25.7.2 Methods

This interface defines a single method that generates the tool tip text:

`public String generateToolTip(PieDataset data, Object category);`
This method is called whenever the `PiePlot` needs to generate a tool tip. It should return a `String` that will be used as the tool tip text.

25.7.3 Notes

A standard implementation (`StandardPieToolTipGenerator`) of this interface is provided with JFreeChart.

You can develop your own tooltip generator, register it with a `PiePlot`, and take full control over the tool tip text that is generated.

See section 11 for information about using tool tips with JFreeChart.

See Also
`StandardPieToolTipGenerator`.

25.8 StandardCategoryToolTipGenerator

25.8.1 Overview

A standard implementation of the `CategoryToolTipGenerator` interface. This class generates tool tips in the format:

```
<series-name>, <category-name> = <value>
```

The value can be formatted as a number or a date, depending on the constructor used to create the generator.

25.8.2 Constructors

To create a generator that formats values as numbers:

`public StandardCategoryToolTipGenerator(NumberFormat formatter);`
Creates a tool tip generator that formats values using the supplied formatter.

To create a generator that formats values as dates (interpreting the numerical value as milliseconds since 1-Jan-1970, in the same way as `java.util.Date`):

`public StandardCategoryToolTipGenerator(DateFormat formatter);`
Creates a tool tip generator that formats values as dates using the supplied formatter.

25.8.3 Notes

Section 11 contains information about using tool tips with JFreeChart.

See Also
`CategoryToolTipGenerator`.

25.9 StandardContourToolTipGenerator

25.9.1 Overview

Not yet documented.

25.10 StandardPieToolTipGenerator

25.10.1 Overview

A standard implementation of the `PieToolTipGenerator` interface. This class generates tool tips in the form:

```
<category> = <number>
```

A `NumberFormat` instance is used to format the data values.

25.10.2 Constructors

The default constructor uses a number formatter for the default locale:

```
public StandardPieToolTipGenerator();
```
Creates a default tool tip generator.

Alternatively, you can supply your own number formatter:

```
public StandardPieToolTipGenerator(NumberFormat formatter);
```
Creates a tool tip generator that uses the specified number formatter.

25.10.3 Notes

Section 11 contains information about using tool tips with JFreeChart.

See Also
`PieToolTipGenerator`.

25.11 StandardXYToolTipGenerator

25.11.1 Overview

A standard implementation of the `XYToolTipGenerator` interface. This class generates tool tips in the format:

```
<series-name> :  x:  <x-value>, y:  <y-value>
```

25.11.2 Constructors

To create a tool tip generator:

```
public StandardXYToolTipGenerator(NumberFormat xFormat,
NumberFormat yFormat);
```
Creates a tool tip generator that uses the supplied number formatters for the x and y values.

25.11.3 Notes

Section **??** contains information about using tool tips with JFreeChart.

See Also
`XYToolTipGenerator`.

25.12 StandardXYZToolTipGenerator

25.12.1 Overview

Not yet documented.

25.13 SymbolicXYToolTipGenerator

25.13.1 Overview

Not yet documented.

25.14 TimeSeriesToolTipGenerator

25.14.1 Overview

Not yet documented.

25.15 ToolTipGenerator

25.15.1 Overview

The base interface for all *tool tip generators*. This interface does not define any methods or attributes.

25.15.2 Notes

Extensions of this interface include:

- `CategoryToolTipGenerator`
- `PieToolTipGenerator`
- `XYToolTipGenerator`

Section 11 contains information about using tool tips with JFreeChart.

25.16 XYToolTipGenerator

25.16.1 Overview

The interface that must be implemented by an *XY tool tip generator*, a class used to generate tool tip text for an `XYPlot`.

25.16.2 Methods

This interface defines a single method:

> `public String generateToolTip(XYDataset data, int series, int item);`
> This method is called whenever the plot needs to generate a tool tip. It should return the tool tip text (which can be anything you want to make it).

25.16.3 Notes

To "install" a tool tip generator, use the `setToolTipGenerator(...)` method in the `AbstractXYItemRenderer` class.

`StandardXYToolTipGenerator` implements this interface, but you are free to write your own implementation to suit your requirements.

Section 11 contains information about using tool tips with JFreeChart.

See Also

`StandardXYToolTipGenerator`.

25.17 XYZToolTipGenerator

25.17.1 Overview

Not yet documented.

26 Package: org.jfree.chart.ui

26.1 Introduction

This package contains user interface classes that can be used to modify chart properties. These classes are optional—they are used in the demonstration application, but you do not need to include this package in your own projects if you do not want to.

26.2 AxisPropertyEditPanel

26.2.1 Overview

Not yet documented.

26.2.2 Notes

Refer to Javadoc HTML files and source code for details.

26.3 ChartPropertyEditPanel

26.3.1 Overview

A panel that displays all the properties of a chart, and allows the user to edit the properties. The panel uses a `JTabbedPane` to display four sub-panels: a `TitlePropertyPanel`, a `LegendPropertyPanel`, a `PlotPropertyPanel` and a panel containing "other" properties (such as the anti-alias setting and the background paint for the chart).

The constructors for this class require a reference to a `Dialog` or a `Frame`. Whichever one is specified is passed on to the `TitlePropertyPanel` and is used if and when a sub-dialog is required for editing titles.

26.3.2 Notes

Refer to Javadoc HTML files and source code for details.

26.4 ColorBarPropertyEditPanel

26.4.1 Overview

Not yet documented.

26.5 ColorPalette

26.5.1 Overview

Not yet documented.

26.6 GreyPalette

26.6.1 Overview

Not yet documented.

26.7 LegendPropertyEditPanel

26.7.1 Overview

Not yet documented.

26.7.2 Notes

Refer to Javadoc HTML files and source code for details.

26.8 NumberAxisPropertyEditPanel

26.8.1 Overview

Not yet documented.

26.8.2 Notes

Refer to Javadoc HTML files and source code for details.

26.9 PaletteChooserPanel

26.9.1 Overview

Not yet documented.

26.10 PlotPropertyEditPanel

26.10.1 Overview

Not yet documented.

26.10.2 Notes

Refer to Javadoc HTML files and source code for details.

26.11 RainbowPalette

26.11.1 Overview

Not yet documented.

26.12 TitlePropertyEditPanel

26.12.1 Overview

Not yet documented.

26.12.2 Notes

Refer to Javadoc HTML files and source code for details.

27 Package: org.jfree.chart.urls

27.1 Overview

This package contains support for URL generation for image maps.

27.2 CategoryURLGenerator

27.2.1 Overview

Not yet documented.

27.3 CustomXYURLGenerator

27.3.1 Overview

Not yet documented.

27.4 PieURLGenerator

27.4.1 Overview

Not yet documented.

27.5 StandardCategoryURLGenerator

27.5.1 Overview

Not yet documented.

27.6 StandardPieURLGenerator

27.6.1 Overview

Not yet documented.

27.7 StandardXYURLGenerator

27.7.1 Overview

Not yet documented.

27.8 StandardXYZURLGenerator

27.8.1 Overview

Not yet documented.

27.9 TimeSeriesURLGenerator

27.9.1 Overview

Not yet documented.

27.10 URLGenerator

27.10.1 Overview

Not yet documented.

27.11 XYURLGenerator

27.11.1 Overview

Not yet documented.

27.12 XYZURLGenerator

27.12.1 Overview

Not yet documented.

28 Package: org.jfree.data

28.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

A design principle in JFreeChart is that there should be a clear separation between the *data* (as represented by the classes in this package) and its *presentation* (controlled by the plot and renderer classes defined elsewhere). For this reason, you will not find methods or attributes that relate to presentation (for example, series colors or line styles) in the dataset classes.

28.2 AbstractDataset

28.2.1 Overview

A useful base class for implementing the `Dataset` interface (or extensions). This class provides a default implementation of the *change listener* mechanism.

28.2.2 Constructors

The default constructor:

> `protected AbstractDataset();`
> Allocates storage for the registered change listeners.

28.2.3 Methods

To register a change listener:

> `public void addChangeListener(DatasetChangeListener listener);`
> Registers a change listener with the dataset. The listener will be notified whenever the dataset changes, via a call to the `datasetChanged(...)` method.

To deregister a change listener:

> `public void removeChangeListener(DatasetChangeListener listener);`
> Deregisters a change listener. The listener will be no longer be notified whenever the dataset changes.

28.2.4 Notes

In most cases, JFreeChart will automatically register listeners for you, and update charts whenever the data changes.

You can implement a dataset without subclassing `AbstractDataset`. This class is provided simply for convenience to save you having to implement your own change listener mechanism.

If you write your own class that extends `AbstractDataset`, you need to remember to call `fireDatasetChanged()` whenever the data in your class is modified.

See Also

`Dataset`, `DatasetChangeListener`, `AbstractSeriesDataset`.

28.3 AbstractSeriesDataset

28.3.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

28.3.2 Constructors

This class is never instantiated directly, so the constructor is protected:

`protected AbstractSeriesDataset();`
Simply calls the constructor of the superclass.

28.3.3 Methods

This method receives series change notifications:

`public void seriesChanged(SeriesChangeEvent event);`
The default behaviour provided by this method is to raise a `DatasetChangeEvent` every time this method is called.

28.3.4 Notes

This class is provided simply for convenience, you are not required to use it when developing your own dataset classes.

See Also
`Dataset`.

28.4 CategoryDataset

28.4.1 Overview

A *category dataset* is a table of values that can be accessed using row and column keys. This type of dataset is most commonly used to create bar charts.

This interface provides the methods required for *reading* the dataset, not for updating it. Classes that implement this interface may be "read-only", or they may provide "write" access.

28.4.2 Methods

This interface adds no additional methods to those defined in the `KeyedValues2D` and `Dataset` interfaces.

28.4.3 Notes

This interface extends the `KeyedValues2D` and `Dataset` interfaces.

The `DefaultCategoryDataset` class provides one implementation of this interface.

The `CategoryToPieDataset` class converts one row or column of the dataset into a `PieDataset`.

See Also

CategoryPlot.

28.5 CategoryToPieDataset

28.5.1 Overview

A utility class that "converts" the data from one row or column of a CategoryDataset into a PieDataset.

28.6 CombinationDataset

28.6.1 Overview

An interface that defines the methods that should be implemented by a *combination dataset*.

28.6.2 Notes

This interface is implemented by the CombinedDataset class.

See Also

CombinedDataset.

28.7 CombinedDataset

28.7.1 Overview

A dataset that can combine other datasets.

28.7.2 Notes

The combined charts feature, originally developed by Bill Kelemen, has been restructured so that it is no longer necessary to use this class. However, you can still use this class if you need to construct a dataset that is the union of existing datasets.

See Also

CombinationDataset.

28.8 ContourDataset

28.8.1 Overview

The dataset used by the ContourPlot class.

See Also

DefaultContourDataset.

28.9 Dataset

28.9.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by `PieDataset`, `CategoryDataset` and `SeriesDataset`.

28.9.2 Methods

This base interface defines two methods for registering change listeners:

`public void addChangeListener(DatasetChangeListener listener);`
Registers a change listener with the dataset. The listener will be notified whenever the dataset changes.

`public void removeChangeListener(DatasetChangeListener listener);`
Deregisters a change listener.

28.9.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as `PieDataset`, `CategoryDataset` or `XYDataset`.

See Also
PieDataset, CategoryDataset, SeriesDataset.

28.10 DatasetChangeEvent

28.10.1 Overview

An event that is used to provide information about changes to datasets.

28.10.2 Constructors

The standard constructor:

`public DatasetChangeEvent(Object source, Dataset dataset);`
Creates a new event. Usually the `source` is the dataset, but this is not guaranteed.

28.10.3 Methods

To get a reference to the `Dataset` that generated the event:

`public Dataset getDataset();`
Returns the dataset which generated the event.

28.10.4 Notes

The current implementation simply indicates that some change has been made to the dataset. In the future, this class may carry more information about the change.

See Also
DatasetChangeListener.

28.11 DatasetChangeListener

28.11.1 Overview

An interface through which dataset change event notifications are posted. If a class needs to receive notification of changes to a dataset, then it should implement this interface and register itself with the dataset.

28.11.2 Methods

The interface defines a single method:

```
public void datasetChanged(DatasetChangeEvent event);
```
Receives notification of a change to a dataset.

28.11.3 Notes

In JFreeChart, the `Plot` class implements this interface in order to receive notification of changes to the dataset.

See Also
DatasetChangeEvent.

28.12 DatasetGroup

28.12.1 Overview

A *dataset group* provides a mechanism for grouping related datasets. At present, this is not used, but in the future it is likely that thread synchronisation will be added to JFreeChart using dataset groups.

28.13 DatasetUtilities

28.13.1 Overview

A collection of utility methods for working with datasets.

28.13.2 Maximum and Minimum Values

To get the minimum domain value in a dataset:

```
public static Number getMinimumDomainValue(Dataset data);
```
Returns the minimum domain value for the dataset. If the dataset implements the DomainInfo interface, then this will be used to obtain the minimum domain value. Otherwise, this method iterates through all of the data.

To get the maximum domain value in a dataset:

```
public static Number getMaximumDomainValue(Dataset data);
```
Returns the maximum domain value for the dataset. If the dataset implements the DomainInfo interface, then this will be used to obtain the maximum domain value. Otherwise, this method iterates through all of the data.

To get the minimum range value in a dataset:

```
public static Number getMinimumRangeValue(Dataset data);
```
Returns the minimum range value for the dataset. If the dataset implements the RangeInfo interface, then this will be used to obtain the minimum range value. Otherwise, this method iterates through all of the data.

To get the maximum range value in a dataset:

```
public static Number getMaximumRangeValue(Dataset data);
```
Returns the maximum range value for the dataset. If the dataset implements the RangeInfo interface, then this will be used to obtain the maximum range value. Otherwise, this method iterates through all of the data.

28.13.3 Creating Datasets

To create a PieDataset from the data in one category of a CategoryDataset:

```
public static PieDataset createPieDataset(CategoryDataset data,
Object category);
```
Returns a pie dataset by taking all the values in the category dataset for the specified category.

To create a PieDataset from the data in one series of a CategoryDataset:

```
public static PieDataset createPieDataset(CategoryDataset data,
int series);
```
Returns a pie dataset by taking all the values in the category dataset for the specified series.

To create an XYDataset by sampling values from a Function2D:

```
public static XYDataset sampleFunction2D(Function2D f,
double start, double end, int samples, String seriesName);
```
Creates a new XYDataset by sampling values in a specified range for the Function2D.

See Also

DomainInfo, RangeInfo.

28.14 DataUtilities

28.14.1 Overview

Not yet documented.

28.15 DateRange

28.15.1 Overview

A *date range*—extends `Range`.

28.16 DefaultCategoryDataset

28.16.1 Overview

A default implementation of the `CategoryDataset` interface.

28.16.2 Constructors

The default constructor creates a new, empty dataset:

> `public DefaultCategoryDataset();`
> Creates a new dataset.

The `DatasetUtilities` class has static methods for creating instances of this class using array data.

28.16.3 Methods

To add a value to the dataset:

> `public addValue(Number value, Comparable rowKey, Comparable columnKey)`
> Adds a value to the dataset. The value can be `null` (to indicate missing data). If there is already a value for the given keys, it is overwritten.

A similar method accepts a `double` value and converts it to a `Number` object before storing it.

Identical `setValue(...)` methods are also provided. These function in exactly the same way as the `addValue(...)` methods.

28.16.4 Notes

This class uses an instance of `DefaultKeyedValues2D` to store its data.

28.17 DefaultContourDataset

28.17.1 Overview

A default implementation of the `ContourDataset` interface.

28.18 DefaultHighLowDataset

28.18.1 Overview

A default implementation of the `HighLowDataset` interface.

28.19 DefaultIntervalCategoryDataset

28.19.1 Overview

A default implementation of the `IntervalCategoryDataset` interface.

28.20 DefaultKeyedValue

28.20.1 Overview

A storage structure for a value that is associated with a key. This class provides a default implementation of the `KeyedValue` interface.

28.21 DefaultKeyedValueDataset

28.21.1 Overview

Not yet documented.

28.22 DefaultKeyedValues

28.22.1 Overview

A storage structure for a collection of values that are associated with keys. This class provides a default implementation of the `KeyedValues` interface.

28.22.2 Notes

The `DefaultPieDataset` class uses an instance of this class to store its data.

28.23 DefaultKeyedValuesDataset

28.23.1 Overview

Not yet documented.

28.24 DefaultKeyedValues2D

28.24.1 Overview

A storage structure for a table of values that are associated with keys. This class provides a default implementation of the `KeyedValues2D` interface.

28.24.2 Notes

The `DefaultCategoryDataset` class uses an instance of this class to store its data.

28.25 DefaultKeyedValues2DDataset

28.25.1 Overview

Not yet documented.

28.26 DefaultMeterDataset

28.26.1 Overview

A default implementation of the `MeterDataset` interface.

28.27 DefaultPieDataset

28.27.1 Overview

A default implementation of the `PieDataset` interface.

28.27.2 Constructors

To create a new pie dataset:

```
public DefaultPieDataset();
```
Creates a new dataset, initially empty.

28.27.3 Methods

To get the value associated with a key:

```
public Number getValue(Comparable key);
```
Returns the value associated with a key (possibly `null`)

To set the value associated with a key:

```
public void setValue(Comparable key, Number value);
```
Sets the value associated with a key.

28.27.4 Notes

The dataset can contain `null` values.

See Also
`PiePlot`.

28.28 DefaultStatisticalCategoryDataset

28.28.1 Overview

A default implementation of the `StatisticalCategoryDataset` interface.

28.29 DefaultValueDataset

28.29.1 Overview

Not yet documented.

28.30 DefaultWindDataset

28.30.1 Overview

A default implementation of the `WindDataset` interface.

28.31 DomainInfo

28.31.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's domain.

28.31.2 Methods

To get the minimum value in the dataset's domain:

```
public Number getMinimumDomainValue();
```
Returns the minimum value in the dataset's domain.

To get the maximum value in the dataset's domain:

```
public Number getMaximumDomainValue();
```
Returns the maximum value in the dataset's domain.

To get the range of values in the dataset's domain:

```
public Range getDomainRange();
```
Returns the range of values in the dataset's domain.

28.31.3 Notes

It is not mandatory for a dataset to implement this interface. However, sometimes it is necessary to calculate the minimum and maximum values in a dataset. Without knowing the internal structure of a dataset, the only means of determining this information is iteration over the entire dataset. If there is a more efficient way to determine the values for your data structures, then you can implement this interface and provide the values directly.

See Also

RangeInfo, DatasetUtilities.

28.32 Function2D

28.32.1 Overview

A simple interface for a 2D function. Implementations of this interface include:

- LineFunction2D;
- PowerFunction2D.

It is a simple matter to implement your own functions.

28.32.2 Methods

The interface defines a single method for obtaining the value of the function for a given input:

```
public double getValue(double x);
```
Returns the value of the function for a given input.

28.32.3 Notes

The `DatasetUtilities` class provides a method for creating an `XYDataset` by sampling the values of a function.

See Also

`LineFunction2D`, `PowerFunction2D`.

28.33 HighLowDataset

28.33.1 Overview

A dataset that supplies data in the form of *high-low-open-close* items. These typically relate to trading data (prices or rates) in financial markets: the open and close values represent the prices at the opening and closing of the trading period, while the high and low values represent the highest and lowest price during the trading period.

Another value returned by this dataset is the *volume*. This represents the volume of trading, and is usually the number of units of the commodity traded during a period. If this data is not available, `null` is returned.

This interface is an extension of the `XYDataset` interface.

28.33.2 Methods

To get the *high* value:

```
public Number getHighValue(int series, int item);
```
Returns the high value for an item within a series.

To get the *low* value:

```
public Number getLowValue(int series, int item);
```
Returns the low value for an item within a series.

To get the *open* value:

```
public Number getOpenValue(int series, int item);
```
Returns the open value for an item within a series.

To get the *close* value:

```
public Number getCloseValue(int series, int item);
```
Returns the close value for an item within a series.

To get the *volume*:

```
public Number getVolumeValue(int series, int item);
```
Returns the volume value for an item within a series.

28.33.3 Notes

This dataset is implemented by the `DefaultHighLowDataset` class, and used by the `CandlestickRenderer` class.

See Also

XYDataset, DefaultHighLowDataset.

28.34 IntervalCategoryDataset

28.34.1 Overview

An extension of the CategoryDataset interface. Methods are added for returning start and end values for a given series and category.

28.34.2 Methods

To get the start value for a series and category:

```
public Number getStartValue(int series, Object category);
```
Returns the start value for the interval.

To get the end value for a series and category:

```
public Number getEndValue(int series, Object category);
```
Returns the end value for the interval.

28.34.3 Notes

The DefaultIntervalCategoryDataset class implements this interface.

See Also:

CategoryDataset, HorizontalIntervalBarRenderer.

28.35 IntervalXYDataset

28.35.1 Overview

A dataset that returns an interval for each of the x and y dimensions. Extends the XYDataset interface.

28.35.2 Methods

To get the start value of the x-interval:

```
public Number getStartXValue(int series, int item);
```
Returns the starting x-value for an item within a series.

To get the end value of the x-interval:

```
public Number getEndXValue(int series, int item);
```
Returns the ending x-value for an item within a series.

To get the start value of the y-interval:

```
public Number getStartYValue(int series, int item);
```
Returns the starting y-value for an item within a series.

To get the end value of the y-interval:

```
public Number getEndYValue(int series, int item);
```
Returns the ending y-value for an item within a series.

28.35.3 Notes

The `TimeSeriesCollection` class implements this interface.

See Also:
`XYDataset`, `IntervalXYZDataset`.

28.36 IntervalXYZDataset

28.36.1 Overview

A natural extension of the `IntervalXYDataset` interface.

28.36.2 Notes

There are no classes that implement this interface at present.

See Also:
`XYDataset`, `IntervalXYDataset`.

28.37 JDBCCategoryDataset

28.37.1 Overview

A *category dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

28.37.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

> `public JDBCCategoryDataset(String url, String driverName, String userName, String password);`
> Creates an empty dataset (no query has been executed yet) and establishes a database connection.

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

> `public JDBCCategoryDataset(Connection con);`
> Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.

If you want to initialise the data via the constructor, rather than creating an empty dataset:

> `public JDBCCategoryDataset(Connection con, String query);`
> Creates a dataset with a pre-existing database connection and executes the specified query.

28.37.3 Methods

This class implements all the methods in the `CategoryDataset` interface (by inheriting them from `DefaultCategoryDataset`).

To refresh the data in the dataset, you need to execute a query against the database:

> `public void executeQuery(String query);`
> Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns at least two columns, the first containing `VARCHAR` data representing categories, and the remaining columns containing numerical data.

You can re-execute the query at any time.

28.37.4 Notes

There is a demo application `JDBCCategoryChartDemo` in the JFreeChart distribution (0.9.3 or later) that illustrates the use of this class.

See Also

`CategoryDataset`, `DefaultCategoryDataset`.

28.38 JDBCPieDataset

28.38.1 Overview

A *pie dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

28.38.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

> `public JDBCPieDataset(String url, String driverName, String userName, String password);`
> Creates an empty dataset (no query has been executed yet) and establishes a database connection.

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

> `public JDBCPieDataset(Connection con);`
> Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.

If you want to initialise the data via the constructor, rather than creating an empty dataset:

> `public JDBCPieDataset(Connection con, String query);`
> Creates a dataset with a pre-existing database connection and executes the specified query.

28.38.3 Methods

This class implements all the methods in the `PieDataset` interface (by inheriting them from `DefaultPieDataset`).

To refresh the data in the dataset, you need to execute a query against the database:

> `public void executeQuery(String query);`
> Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns two columns, the first containing `VARCHAR` data representing categories, and the second containing numerical data.

You can re-execute the query at any time.

28.38.4 Notes

There is a demo application `JDBCPieChartDemo` in the JFreeChart distribution (0.9.3 or later) that illustrates the use of this class.

See Also

`PieDataset`, `DefaultPieDataset`.

28.39 JDBCXYDataset

28.39.1 Overview

An *XY dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

28.39.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

> `public JDBCXYDataset(String url, String driverName, String userName, String password);`
> Creates an empty dataset (no query has been executed yet) and establishes a database connection.

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

> `public JDBCXYDataset(Connection con);`
> Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.

If you want to initialise the data via the constructor, rather than creating an empty dataset:

> `public JDBCXYDataset(Connection con, String query);`
> Creates a dataset with a pre-existing database connection and executes the specified query.

28.39.3 Methods

This class implements all the methods in the `XYDataset` interface.

To refresh the data in the dataset, you need to execute a query against the database:

> `public void executeQuery(String query);`
> Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns at least two columns, the first containing numerical or date data representing x-values, and the remaining column(s) containing numerical data for each series (one series per column).

You can re-execute the query at any time.

28.39.4 Notes

There is a demo application `JDBCXYChartDemo` in the JFreeChart distribution (0.9.3 or later) that illustrates the use of this class.

See Also
`XYDataset`.

28.40 KeyedObject

28.40.1 Overview

Not yet documented.

28.41 KeyedObjects

28.41.1 Overview

Not yet documented.

28.42 KeyedObjects2D

28.42.1 Overview

Not yet documented.

28.43 KeyedValue

28.43.1 Overview

A *keyed value* is a value that is associated with a key. This interface extends the `Value` interface.

28.43.2 Methods

To access the key associated with the value:

> `public Comparable getKey();`
> Returns the key associated with the value.

28.43.3 Notes

The `DefaultKeyedValue` class provides one implementation of this interface.

28.44 KeyedValueComparator

28.44.1 Overview

Not yet documented.

28.45 KeyedValueComparatorType

28.45.1 Overview

Not yet documented.

28.46 KeyedValueDataset

28.46.1 Overview

Not yet documented.

28.47 KeyedValues

28.47.1 Overview

A collection of *keyed values* (that is, values that are associated with a key). This interface extends the `Values` interface.

28.47.2 Methods

To access the key associated with a value:

```
public Comparable getKey(int index);
```
Returns the key associated with an item in the collection.

To convert a key into an item index:

```
public int getIndex(Comparable key);
```
Returns the item index for a key.

To get a list of all keys in the collection:

```
public List getKeys();
```
Returns a list of the keys in the collection.

To get the value associated with a key:

```
public Number getValue(Comparable key);
```
Returns the value associated with a key.

28.47.3 Notes

The `DefaultKeyedValues` class provides one implementation of this interface.

28.48 KeyedValuesDataset

28.48.1 Overview

A *keyed values dataset* is a collection of values where each value is associated with a key. A common use for this type of dataset is in the creation of pie charts.

28.48.2 Methods

This interface adds no methods to those it inherits from the `KeyedValues` and `Dataset` interfaces.

28.49 KeyedValues2D

28.49.1 Overview

A table of values that can be accessed using a *row key* and a *column key*. This interface extends the `Values2D` interface.

28.49.2 Methods

To get the key for a row:

```
public Comparable getRowKey(int row);
```
Returns the key associated with a row.

To convert a row key into an index:

```
public int getRowIndex(Comparable key);
```
Returns the row index for the given key.

To get a list of the row keys:

```
public List getRowKeys();
```
Returns a list of the row keys.

To get the key for a column:

```
public Comparable getColumnKey(int column);
```
Returns the key associated with a column.

To convert a column key into an index:

```
public int getColumnIndex(Comparable key);
```
Returns the column index for a given key.

To return a list of column keys:

```
public List getColumnKeys();
```
Returns a list of the column keys.

To get the value associated with a pair of keys:

```
public Number getValue(Comparable rowKey, Comparable columnKey);
```
Returns the value associated with the keys.

28.49.3 Notes

The `DefaultKeyedValues2D` class provides one implementation of this interface.

28.50 KeyedValues2DDataset

28.50.1 Overview

Not yet documented.

28.51 LineFunction2D

28.51.1 Overview

A simple function of the form $y = a + bx$.

28.51.2 Constructor

To construct a new line function:

```
public LineFunction2D(double a, double b);
```
Creates a new line function with the given coefficients.

28.51.3 Methods

```
public double getValue(double x);
```
Returns the value of the function for a given input.

28.51.4 Notes

This class implements the `Function2D` interface.

The `RegressionDemo1` application provides an example of this class being used.

See Also

`PowerFunction2D`.

28.52 MeanAndStandardDeviation

28.52.1 Overview

Not yet documented.

28.53 MeterDataset

28.53.1 Overview

A dataset that supplies a single value within some overall range. In addition, the dataset defines three subranges: a "normal" range, a "warning" range, and a "critical" range.

This dataset can be used to display meters and gauges. The normal, warning, and critical ranges can be used to color code a meter or gauge and provide context for the meter reading.

28.53.2 Methods

To get the current value (or meter reading):

`public Number getValue();`
Returns the current value.

To get the overall range:

`public Number getMinimumValue();`
Returns the lowest possible value.

`public Number getMaximumValue();`
Returns the highest possible value.

To get the "normal" range (a subset of the overall range):

`public Number getMinimumNormalValue();`
Returns the lower bound of the "normal" range.

`public Number getMaximumNormalValue();`
Returns the upper bound of the "normal" range.

To get the "warning" range (a subset of the overall range):

`public Number getMinimumWarningValue();`
Returns the lower bound of the "warning" range.

`public Number getMaximumWarningValue();`
Returns the upper bound of the "warning" range.

To get the "critical" range (a subset of the overall range):

28.53.3 Notes

The `DefaultMeterDataset` class provides one implementation of this interface.

There is an argument for moving the "normal", "warning" and "critical" range settings to the plot classes, since they relate to the *presentation* of the data, rather than being part of the data itself. I've chosen (for now at least) to leave the code in the form that it was contributed.

See Also:
DefaultMeterDataset, MeterPlot.

28.54 MovingAverage

28.54.1 Overview

A utility class for calculating moving average series.

28.54.2 Methods

To calculate the moving average of a time series:

> `public static TimeSeries createMovingAverage(TimeSeries source, String name, int periodCount, int skip);`
> Creates a new series containing moving average values based on the `source` series. The new series will be called `name`. The `periodCount` specifies the number of periods over which the average is calculated, and `skip` controls the initial number of periods for which no average is calculated (usually the same as the `periodCount`).

28.54.3 Notes

The `MovingAverageDemo` class in the JFreeChart distribution provides one example of how to use this class.

28.55 MultiIntervalCategoryDataset

28.55.1 Overview

An extension of the `IntervalCategoryDataset` interface that allows multiple intervals for each category.

28.55.2 Notes

The `TaskSeriesCollection` class implements this interface.

28.56 NonGridContourDataset

28.56.1 Overview

Not yet documented.

28.57 PieDataset

28.57.1 Overview

A *pie dataset* is a collection of values where each value is associated with a key. This type of dataset is most commonly used to create pie charts.

28.57.2 Methods

This interface adds no methods to those it inherits from the `KeyedValuesDataset` interface.

28.57.3 Notes

The `DefaultPieDataset` class provides one implementation of this interface.

The `DatasetUtilities` class includes some methods for creating a `PieDataset` by slicing a `CategoryDataset` either by row or column.

See Also

`CategoryToPieDataset`, `PiePlot`.

28.58 PowerFunction2D

28.58.1 Overview

A function of the form $y = ax^b$.

28.58.2 Constructor

To construct a new power function:

```
public PowerFunction2D(double a, double b);
```
Creates a new power function with the given coefficients.

28.58.3 Methods

```
public double getValue(double x);
```
Returns the value of the function for a given input.

28.58.4 Notes

This class implements the `Function2D` interface.

The `RegressionDemo1` application provides an example of this class being used.

See Also

`LineFunction2D`.

28.59 Range

28.59.1 Overview

Represents a range of values.

28.59.2 Methods

To get the *lower bound* of the range:

```
public double getLowerBound();
```
Returns the lower bound for the range.

To get the *upper bound* of the range:

```
public double getUpperBound();
```
Returns the upper bound for the range.

28.59.3 Notes

The DateRange class extends this class to support a date range.

28.60 RangeInfo

28.60.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's range.

28.60.2 Methods

To get the minimum value in the dataset's range:

```
public Number getMinimumRangeValue();
```
Returns the minimum value in the dataset's range.

To get the maximum value in the dataset's range:

```
public Number getMaximumRangeValue();
```
Returns the maximum value in the dataset's range.

To get the range of values in the dataset's range:

```
public Range getValueRange();
```
Returns the range of values in the dataset's range.

28.60.3 Notes

It is not mandatory for a dataset to implement this interface. However, sometimes it is necessary to calculate the minimum and maximum values in a dataset. Without knowing the internal structure of a dataset, the only means of determining this information is iteration over the entire dataset. If there is a more efficient way to determine the values for your data structures, then you can implement this interface and provide the values directly.

See Also

DomainInfo.

28.61 Regression

28.61.1 Overview

This class provides some utility methods for calculating regression co-efficients. In version 0.9.4 of JFreeChart, two regression types are supported:

- linear (OLS) regression;
- power regression.

28.61.2 Methods

To calculate the OLS regression for an array of data values:

```
public static double[] getOLSRegression(double[][] data);
```
Performs an ordinary least squares regression on the data.

28.62 Series

28.62.1 Overview

A useful base class for implementing data series. Subclasses include `TimeSeries` and `XYSeries`.

28.62.2 Constructor

The constructor is `protected` since you do not create a `Series` directly, but via a subclass:

```
protected Series(String name, String description);
```
Creates a new series.

See Also

`AbstractSeriesDataset`, `TimeSeries`.

28.63 SeriesChangeEvent

28.63.1 Overview

An event class that is passed to a `SeriesChangeListener` to notify it concerning a change to a series.

28.64 SeriesChangeListener

28.64.1 Overview

The interface through which series change notifications are posted.

Typically a dataset will implement this interface to receive notification of any changes to the individual series in the dataset (which will normally be passed on as a `DatasetChangeEvent`).

28.64.2 Methods

This interface defines a single method:

```
public void seriesChanged(SeriesChangeEvent event);
```
Receives notification when a series changes.

28.64.3 Notes

The `AbstractSeriesDataset` class implements this interface, generating a `DatasetChangeEvent` every time it receives notification of a `SeriesChangeEvent`.

28.65 SeriesDataset

28.65.1 Overview

A base interface that defines a dataset containing zero, one or many data series.

28.65.2 Methods

To find out how many series there are in a dataset:

```
public int getSeriesCount();
```
Returns the number of series in the dataset.

To get the name of a series:

```
public String getSeriesName(int series);
```
Returns the name of the series with the specified index (zero based).

28.65.3 Notes

This interface is extended by `CategoryDataset` and `XYDataset`.

See Also:
`CategoryDataset`, `XYDataset`.

28.66 SeriesException

28.66.1 Overview

A general exception that can be thrown by a `Series`.

For example, a time series will not allow duplicate time periods—attempting to add a duplicate time period will throw a `SeriesException`.

28.67 SignalsDataset

28.67.1 Overview

Not yet documented.

28.68 SortOrder

28.68.1 Overview

Not yet documented.

28.69 StatisticalCategoryDataset

28.69.1 Overview

A *statistical category dataset* is a table of data where each data item consists of a mean and a standard deviation (calculated externally on the basis of some other data). This interface is an extension of the `CategoryDataset` interface.

28.69.2 Methods

To get the mean value for an item in the dataset, using row and column indices:

```
public Number getMeanValue(int row, int column);
```
Returns the mean value for one cell in the table.

Alternatively, you can access the same value using the row and column keys:

```
public Number getMeanValue(Comparable rowKey, Comparable columnKey);
```
Returns the mean value for one cell in the table.

To get the standard deviation value for an item in the dataset, using row and column indices:

```
public Number getStdDevValue(int row, int column);
```
Returns the standard deviation for one cell in the table.

As with the mean value, you can also access the standard deviation using the row and column keys:

```
public Number getStdDevValue(Comparable rowKey, Comparable columnKey);
```
Returns the standard deviation for one cell in the table.

28.69.3 Notes

The `DefaultStatisticalCategoryDataset` class implements this interface.

28.70 Statistics

28.70.1 Overview

Provides some static utility methods for calculating statistics.

28.70.2 Methods

To calculate the average of an array of `Number` objects:

`public static double getAverage(Number[] data);`
Returns the average of an array of numbers.

To calculate the standard deviation of an array of `Number` objects:

`public static double getStdDev(Number[] data);`
Returns the standard deviation of an array of numbers.

To calculate a least squares regression line through an array of data:

`public static double[] getLinearFit(Number[] x_data, Number[] y_data);`
Returns the intercept (`double[0]`) and slope (`double[1]`) of the linear regression line.

To calculate the slope of a least squares regression line:

`public static double getSlope(Number[] x_data, Number[] y_data);`
Returns the slope of the linear regression line.

To calculate the slope of a least squares regression line:

`public static double getCorrelation(Number[] data1, Number[] data2);`
Returns the correlation between two sets of numbers.

28.70.3 Notes

This class was written by Matthew Wright.

28.71 SubseriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

28.72 Task

28.72.1 Overview

A class that represents a *task*, consisting of:

- a task description;
- a duration (estimated or actual);
- a list of sub-tasks;

In JFreeChart, tasks are used in the construction of *Gantt charts*. One or more related tasks can be added to a `TaskSeries`. In turn, one or more `TaskSeries` can be added to a `TaskSeriesCollection`.

28.73 TaskSeries

28.73.1 Overview

A *task series* is a collection of related tasks.
You can add one or more `TaskSeries` objects to a `TaskSeriesCollection` to create a dataset that can be used to produce *Gantt charts*.

28.74 TaskSeriesCollection

28.74.1 Overview

A *task series collection* contains one or more `TaskSeries` objects, and provides access to the task information via the `MultiIntervalCategoryDataset` interface. You can use this class as the dataset for a *Gantt chart*.

28.75 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

28.76 Value

28.76.1 Overview

An interface for accessing a single value.

28.76.2 Methods

The interface defines a single method for accessing the value:

```
public Number getValue();
```
Returns the value.

28.76.3 Notes

The `KeyedValue` interface extends this interface and the `DefaultKeyedValue` class provides one implementation of this interface.

28.77 ValueDataset

28.77.1 Overview

Not yet documented.

28.78 Values

28.78.1 Overview

An interface for accessing a collection of values.

28.78.2 Methods

To get the number of items in the collection:

```
public int getItemCount();
```
Returns the number of items in the collection.

To get a value from the collection:

```
public Number getValue(int item);
```
Returns a value from the collection (possibly `null`).

28.78.3 Notes

The `KeyedValues` interface extends this interface.

The `DefaultKeyedValues` class provides one implementation of this interface.

28.79 Values2D

28.79.1 Overview

An interface for accessing a table of values.

28.79.2 Methods

To get the number of rows in the table:

`public int getRowCount();`
Returns the row count.

To get the number of columns in the table:

`public int getColumnCount();`
Returns the column count.

To get a value from one cell in the table:

`public Number getValue(int row, int column);`
Returns a value (possibly `null`) from a cell in the table.

28.79.3 Notes

The `KeyedValues2D` interface extends this interface.

The `DefaultKeyedValues2D` class provides one implementation of this interface.

28.80 WindDataset

28.80.1 Overview

Not yet documented.

28.81 XYDatapair

28.81.1 Overview

Associates a numerical value with another numerical value. This class parallels the `TimeSeriesDataPair` class.

28.82 XYDataset

28.82.1 Overview

An interface that defines a collection of data in the form of *(x, y)* values. The dataset can consist of zero, one or many data series. The *(x, y)* values in one series are completely independent of the *(x, y)* value in the other series in the dataset (that is, x-values are not "shared").

Extensions of this interface include: `IntervalXYDataset`, `HighLowDataset` and `XYZDataset`.

28.82.2 Methods

To get the number of items in a series:

```
public int getItemCount(int series);
```
Returns the number of data items in a series.

To get the *x-value* for an item within a series:

```
public Number getXValue(int series, int item);
```
Returns an x-value for a series.

To get the *y-value* for an item within a series:

```
public Number getYValue(int series, int item);
```
Returns a y-value for a series (possibly `null`).

28.82.3 Notes

It is often pointed out to me that using `double` values instead of `Number` objects would speed up the access to data. That is true, but I have decided to stick with using `Number` objects for two reasons:

- it allows `null` to be used to indicate an unknown data value;
- objects can be more conveniently displayed using standard Java components such as Swing's `JTable`.

See Also:
`SeriesDataset`, `IntervalXYDataset`.

28.83 XYSeries

28.83.1 Overview

A series of (x, y) data items (extends `Series`). Each item is represented by an instance of `XYDataPair` and stored in a list (sorted in ascending order of x-values).

From version 0.9.3 onwards, `XYSeries` will allow duplicate x-values. There is, however, an option to create an instance of this class that does not allow duplicate x-values.

28.83.2 Constructors

To construct a series:

`public XYSeries(String name);`
Creates a new series (initially empty) with the specified name. Duplicate x-values will be allowed.

To construct a series with control over whether or not duplicate x-values are permitted:

`public XYSeries(String name, boolean allowDuplicateXValues);`
Creates a new series (initially empty) with the specified name. Duplicate x-values will be allowed or disallowed, as specified.

28.83.3 Methods

To find out how many items are contained in a series:

`public int getItemCount();`
Returns the number of items in the series.

To add new data to a series:

`public void add(double x, double y);`
Adds a new data item to the series. Note that duplicate x values may not be allowed (refer to the constructor for details).

To update an existing data value:

`public void update(int item, Number y);`
Changes the value of one item in the series. The `item` is a zero-based index.

To clear all values from the series:

`public void clear();`
Clears all values from the series.

28.83.4 Notes

This class extends `Series`, so you can register change listeners with the series.

You can create a collection of series using the `XYSeriesCollection` class. Since `XYSeriesCollection` implements the `XYDataset` interface, this is a convenient structure for supplying data to JFreeChart.

See Also:
`XYSeriesCollection`.

28.84 XYSeriesCollection

28.84.1 Overview

A collection of `XYSeries` objects. This class implements the `XYDataset` interface, so can be used very conveniently with JFreeChart.

28.84.2 Constructors

To construct a series collection:

`public XYSeriesCollection();`
Creates a new empty series collection.

28.84.3 Methods

To add a series to the collection:

`public void addSeries(XYSeries series);`
Adds a series to the collection. Registered listeners are notified that the dataset has changed.

To find out how many series are held in the collection:

`public int getSeriesCount();`
Returns the number of series in the collection.

To access a particular series:

`public XYSeries getSeries(int series);`
Returns a series from the collection. The `series` argument is a zero-based index.

28.84.4 Notes

This class implements the `XYDataset` interface, so it is a convenient class for use with JFreeChart.

See Also:
`XYSeries`.

28.85 XYZDataset

A natural extension of the `XYDataset` interface.

28.86 XisSymbolic

28.86.1 Overview

Not yet documented.

28.87 YisSymbolic

28.87.1 Overview

Not yet documented.

29 Package: org.jfree.data.time

29.1 Introduction

This package contains interfaces and classes that are used to represent *time-based* data. The `TimeSeriesCollection` class is used to store one or more `TimeSeries` objects, and provides an implementation of the `XYDataset` interface (so that it can be displayed using the `XYPlot` class.

29.2 Day

29.2.1 Overview

A *regular time period* that is one day long. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

29.2.2 Usage

A common use for this class is to represent daily data in a time series. For example:

```
TimeSeries series = new TimeSeries("Daily  Data");
series.add(new Day(1, SerialDate.MARCH, 2003), 10.2);
series.add(new Day(3, SerialDate.MARCH, 2003), 17.3);
series.add(new Day(4, SerialDate.MARCH, 2003), 14.6);
series.add(new Day(7, SerialDate.MARCH, 2003), null);
```

Note that the `SerialDate` class is defined in the JCommon class library.

29.2.3 Constructor

There are several different ways to create a new `Day` instance. You can specify the day, month and year:

`public Day(int day, int month, int year);`
Creates a new `Day` instance. The `month` argument should be in the range 1 to 12. The `year` argument should be in the range 1900 to 9999.

You can create a `Day` instance based on a `SerialDate` (defined in the JCommon class library):

`public Day(SerialDate day);`
Creates a new `Day` instance.

You can create a `Day` instance based on a `Date`:

`public Day(Date time);`
Creates a new `Day` instance.

Finally, the default constructor creates a `Day` instance based on the current system date:

`public Day();`
Creates a new `Day` instance for the current system date.

29.2.4 Methods

There are methods to return the year, month and day-of-the-month:

`public int getYear();`
Returns the year (in the range 1900 to 9999).

`public int getMonth();`
Returns the month (in the range 1 to 12).

`public int getDayOfMonth();`
Returns the day-of-the-month (in the range 1 to 31).

There is no method to *set* these attributes, because this class is immutable.

To return a `SerialDate` instance that represents the same day as this object:

`public SerialDate getSerialDate();`
Returns the day as a `SerialDate`.

Given a `Day` object, you can create an instance representing the previous day or the next day:

`public RegularTimePeriod previous();`
Returns the previous day, or `null` if the lower limit of the range is reached.

`public RegularTimePeriod next();`
Returns the next day, or `null` if the upper limit of the range is reached.

To convert a `Day` object to a `String` object:

`public String toString();`
Returns a string representing the day.

To convert a `String` object to a `Day` object:

`public static Day parseDay(String s) throws TimePeriodFormatException;`
Parses the string and, if possible, returns a `Day` object.

29.2.5 Notes

Points to note:

- in the current implementation, the day can be in the range 1-Jan-1900 to 31-Dec-9999.
- the `Day` class is immutable, a requirement for all `RegularTimePeriod` subclasses.

29.3 FixedMillisecond

29.3.1 Overview

A *regular time period* that is one millisecond in length. This class uses the same encoding convention as `java.util.Date`. Unlike the other regular time period classes, `FixedMillisecond` is fixed in real time. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

29.3.2 Constructors

To create a new `FixedMillisecond`:

> `public FixedMillisecond(long millisecond);`
> Creates a new `FixedMillisecond` instance. The `millisecond` argument uses the same encoding as `java.util.Date`.

You can construct a a `FixedMillisecond` instance based on a `java.util.Date` instance:

> `public FixedMillisecond(Date time);`
> Creates a new `FixedMillisecond` instance representing the same millisecond as the `time` argument.

A default constructor is provided, which creates a `FixedMillisecond` instance based on the current system time:

> `public FixedMillisecond();`
> Creates a new `FixedMillisecond` instance based on the current system time.

29.3.3 Methods

Given a `FixedMillisecond` object, you can create an instance representing the previous millisecond:

> `public RegularTimePeriod previous();`
> Returns the previous millisecond, or `null` if the lower limit of the range is reached.

...and the next millisecond:

> `public RegularTimePeriod next();`
> Returns the next millisecond, or `null` if the upper limit of the range is reached.

29.3.4 Notes

Points to note:

- this class is just a wrapper for the `java.util.Date` class, to allow it to be used as a `RegularTimePeriod`.
- the `FixedMillisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.4 Hour

29.4.1 Overview

A *regular time period* one hour in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

29.4.2 Usage

A common use for this class is to represent hourly data in a time series. For example:

```
TimeSeries series = new TimeSeries("Hourly Data", Hour.class);
Day today = new Day();
series.add(new Hour(3, today), 734.4);
series.add(new Hour(4, today), 453.2);
series.add(new Hour(7, today), 500.2);
series.add(new Hour(8, today), null);
series.add(new Hour(12, today), 734.4);
```

Note that the hours in the `TimeSeries` do not have to be consecutive.

29.4.3 Constructor

There are several ways to create a new `Hour` instance. You can specify the hour and day:

`public Hour(int hour, Day day);`
Creates a new `Hour` instance. The `hour` argument should be in the range 0 to 23.

Alternatively, you can supply a `java.util.Date`:

`public Hour(Date time);`
Creates a new `Hour` instance. The default time zone is used to decode the `Date`.

A default constructor is provided:

`public Hour();`
Creates a new `Hour` instance based on the current system time.

29.4.4 Methods

To access the hour and day:

`public int getHour();`
Returns the hour (in the range 0 to 23).

`public Day getDay();`
Returns the day.

There is no method to *set* the hour or the day, because this class is immutable.

Given a `Hour` object, you can create an instance representing the previous hour:

`public RegularTimePeriod previous();`
Returns the previous hour, or `null` if the lower limit of the range is reached.

...or the next hour:

`public RegularTimePeriod next();`
Returns the next hour, or `null` if the upper limit of the range is reached.

29.4.5 Notes

The `Hour` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.5 Millisecond

29.5.1 Overview

A *regular time period* one millisecond in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

29.5.2 Constructors

To construct a `Millisecond` instance:

> `public Millisecond(int millisecond, Second second);`
> Creates a new `Millisecond` instance. The `millisecond` argument should be in the range 0 to 999.

To construct a `Millisecond` instance based on a `java.util.Date`:

> `public Millisecond(Date date);`
> Creates a new `Millisecond` instance.

A default constructor is provided:

> `public Millisecond();`
> Creates a new `Millisecond` instance based on the current system time.

29.5.3 Methods

To access the millisecond:

> `public int getMillisecond();`
> Returns the second (in the range 0 to 999).

To access the `Second`:

> `public Second getSecond();`
> Returns the `Second`.

There is no method to *set* the millisecond or the second, because this class is immutable.

Given a `Millisecond` object, you can create an instance representing the previous millisecond:

> `public RegularTimePeriod previous();`
> Returns the previous millisecond, or `null` if the lower limit of the range is reached.

...or the next:

> `public RegularTimePeriod next();`
> Returns the next millisecond, or `null` if the upper limit of the range is reached.

29.5.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.6 Minute

29.6.1 Overview

A *regular time period* one minute in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations.

29.6.2 Constructors

There are several ways to create new instances of this class. You can specify the minute and hour:

```
public Minute(int minute, Hour hour);
```
Creates a new `Minute` instance. The `minute` argument should be in the range 0 to 59.

Alternatively, you can supply a `java.util.Date`:

```
public Minute(Date time);
```
Creates a new `Minute` instance based on the supplied date/time.

A default constructor is provided:

```
public Minute();
```
Creates a new `Minute` instance, based on the current system time.

29.6.3 Methods

To access the minute and hour:

```
public int getMinute();
```
Returns the minute (in the range 0 to 59).

```
public Hour getHour();
```
Returns the hour.

There is no method to *set* the minute or the day, because this class is immutable.

Given a `Minute` object, you can create an instance representing the previous minute:

```
public RegularTimePeriod previous();
```
Returns the previous minute, or `null` if the lower limit of the range is reached.

...or the next:

```
public RegularTimePeriod next();
```
Returns the next minute, or `null` if the upper limit of the range is reached.

29.6.4 Notes

The `Minute` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.7 Month

29.7.1 Overview

A *time period* representing a month in a particular year. This class is designed to be used with the `TimeSeries` class, but could be used in other contexts as well. Extends `RegularTimePeriod`.

29.7.2 Constructors

There are several ways to create new instances of this class. You can specify the month and year:

`public Month(int month, Year year);`
Creates a new `Month` instance. The `month` argument should be in the range 1 to 12.

`public Month(int month, int year);`
Creates a new `Month` instance. The `month` argument should be in the range 1 to 12. The year argument should be in the range 1900 to 9999.

Alternatively, you can specify a `java.util.Date`:

`public Month(Date time);`
Creates a new `Month` instance.

A default constructor is provided:

`public Month();`
Creates a new `Month` instance, based on the current system time.

29.7.3 Methods

To access the month and year:

`public int getMonth();`
Returns the month (in the range 1 to 12).

`public Year getYear();`
Returns the year.

`public int getYearValue();`
Returns the year as an `int`.

There is no method to *set* the month or the year, because this class is immutable.

Given a `Month` object, you can create an instance representing the previous month:

`public RegularTimePeriod previous();`
Returns the previous month, or `null` if the lower limit of the range is reached.

...or the next month:

`public RegularTimePeriod next();`
Returns the next month, or `null` if the upper limit of the range is reached.

To convert a `Month` object to a `String` object:

`public String toString();`
Returns a string representing the month.

29.7.4 Notes

Points to note:

- the year can be in the range 1900 to 9999.
- this class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.8 Quarter

29.8.1 Overview

A calendar quarter—this class extends `RegularTimePeriod`.

29.8.2 Usage

A common use for this class is representing quarterly data in a time series:

```
TimeSeries series = new TimeSeries("Quarterly Data", Quarter.class);
series.add(new Quarter(1, 2001), 500.2);
series.add(new Quarter(2, 2001), 694.1);
series.add(new Quarter(3, 2001), 734.4);
series.add(new Quarter(4, 2001), 453.2);
series.add(new Quarter(1, 2002), 500.2);
series.add(new Quarter(2, 2002), null);
series.add(new Quarter(3, 2002), 734.4);
series.add(new Quarter(4, 2002), 453.2);
```

29.8.3 Constructor

There are several ways to create a new `Quarter` instance. You can specify the quarter and year:

`public Quarter(int quarter, Year year);`
Creates a new `Quarter` instance. The `quarter` argument should be in the range 1 to 4.

`public Quarter(int quarter, int year);`
Creates a new `Quarter` instance.

Alternatively, you can supply a `java.util.Date`:

`public Quarter(Date time);`
Creates a new `Quarter` instance.

A default constructor is provided:

`public Quarter();`
Creates a new `Quarter` instance based on the current system time.

29.8.4 Methods

To access the quarter and year:

`public int getQuarter();`
Returns the quarter (in the range 1 to 4).

`public Year getYear();`
Returns the year.

There is no method to *set* the quarter or the year, because this class is immutable.

Given a `Quarter` object, you can create an instance representing the previous or next quarter:

`public RegularTimePeriod previous();`
Returns the previous quarter, or `null` if the lower limit of the range is reached.

`public RegularTimePeriod next();`
Returns the next quarter, or `null` if the upper limit of the range is reached.

To convert a `Quarter` object to a `String` object:

`public String toString();`
Returns a string representing the quarter.

29.8.5 Notes

Points to note:

- the year can be in the range 1900 to 9999.
- this class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.9 RegularTimePeriod

29.9.1 Overview

An abstract class that represents a *time period* that occurs at some regular interval. A number of concrete subclasses have been implemented: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

29.9.2 Time Zones

The time periods represented by this class and its subclasses typically "float" with respect to any specific time zone. For example, if you define a `Day` object to represent `1-Apr-2002`, then that is the day it represents *no matter where you are in the world*. Of course, against a real time line, `1-Apr-2002` in (say) New Zealand is not the same as `1-Apr-2002` in (say) France. But *sometimes* you want to treat them as if they were the same, and that is what this class does.[8]

29.9.3 Conversion To/From Date Objects

Occasionally you may want to convert a `RegularTimePeriod` object into an instance of `java.util.Date`. The latter class represents a precise moment in real time (as the number of milliseconds since January 1, 1970, 00:00:00.000 GMT), so to do the conversion you have to "peg" the `RegularTimePeriod` instance to a particular time zone.

The `getStart(...)` and `getEnd(...)` methods provide this facility, using the default timezone. In addition, there are other methods to return the first, last and middle milliseconds for the time period, using the default time zone, a user supplied timezone, or a `Calendar` with the timezone preset.

29.9.4 Methods

Given a `RegularTimePeriod` instance, you can create another instance representing the previous or next time period:

> `public abstract RegularTimePeriod previous();`
> Returns the previous time period, or `null` if the current time period is the first in the supported range.
>
> `public abstract RegularTimePeriod next();`
> Returns the next time period, or `null` if the current time period is the last in the supported range.

To assist in converting the time period to a `java.util.Date` object, the following methods peg the time period to a particular time zone and return the first and last millisecond of the time period (using the same encoding convention as `java.util.Date`):

> `public long getFirstMillisecond();`
> Returns the first millisecond of the time period, evaluated using the default timezone.
>
> `public long getFirstMillisecond(TimeZone zone);`
> Returns the first millisecond of the time period, evaluated using a particular timezone.
>
> `public abstract long getFirstMillisecond(Calendar calendar);`
> Returns the first millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

[8] For example, an accountant might be adding up sales for all the subsidiaries of a multinational company. Sales on `1-Apr-2002` in New Zealand are added to sales on `1-Apr-2002` in France, even though the real time periods are offset from one another.

`public long getMiddleMillisecond();`
Returns the middle millisecond of the time period, evaluated using the default timezone.

`public long getMiddleMillisecond(TimeZone zone);`
Returns the middle millisecond of the time period, evaluated using a particular timezone.

`public long getMiddleMillisecond(Calendar calendar);`
Returns the middle millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

`public long getLastMillisecond();`
The last millisecond of the time period, evaluated using the default timezone.

`public long getLastMillisecond(TimeZone zone);`
Returns the last millisecond of the time period, evaluated using a particular timezone.

`public abstract long getLastMillisecond(Calendar calendar);`
Returns the last millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

29.9.5 Notes

Points to note:

- this class and its subclasses can be used with the `TimeSeries` class.
- all `RegularTimePeriod` subclasses are required to be immutable.
- known subclasses include: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

29.10 Second

29.10.1 Overview

A *regular time period* that is one second long. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

29.10.2 Constructors

There are several ways to create new instances of this class. You can specify the minute and second:

`public Second(int second, Minute minute);`
Creates a new `Second` instance. The `second` argument should be in the range 0 to 59.

Alternatively, you can supply a `java.util.Date`:

`public Second(Date date);`
Creates a new `Second` instance.

A default constructor is provided:

`public Second();`
Creates a new `Second` instance based on the current system time.

29.10.3 Methods

To access the second and minute:

`public int getSecond();`
Returns the second (in the range 0 to 59).

`public Minute getMinute();`
Returns the minute.

There is no method to *set* the second or the minute, because this class is immutable.

Given a `Second` object, you can create an instance representing the previous second or the next second:

`public RegularTimePeriod previous();`
Returns the previous second, or `null` if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next second, or `null` if the upper limit of the range is reached.

29.10.4 Notes

The `Second` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

29.11 SimpleTimePeriod

29.11.1 Overview

A simple implementation of the `TimePeriod` interface.

29.11.2 Methods

To return the start and end dates:

`public Date getStart();`
Returns the start date for the period.

`public Date getEnd();`
Returns the end date for the period.

29.12 TimePeriod

29.12.1 Overview

A period of time defined by two `java.util.Date` instances representing the start and end of the time period.

29.12.2 Methods

To get the start and end of the time period:

`public Date getStart();`
Returns the start of the time period.

`public Date getEnd();`
Returns the end of the time period.

29.12.3 Notes

This interface is implemented by:

- the `SimpleTimePeriod` class;
- the `RegularTimePeriod` base class and all its subclasses.

29.13 TimePeriodFormatException

29.13.1 Overview

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

29.14 TimeSeries

29.14.1 Overview

A time series is a data structure that associates numeric values with particular time periods. In other words, a collection of data values in the form *(timeperiod, value)*.

The time periods are represented by subclasses of `RegularTimePeriod`, including `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

29.14.2 Usage

A time series may contain zero, one or many time periods with associated data values. You can assign a null value to a time period, and you can skip time periods completely. You cannot add duplicate time periods to a time series. Different subclasses of `RegularTimePeriod` cannot be mixed within one time series.

Here is an example showing how to create a series with quarterly data:

```
TimeSeries series = new TimeSeries("Quarterly Data", Quarter.class);
series.add(new Quarter(1, 2001), 500.2);
series.add(new Quarter(2, 2001), 694.1);
series.add(new Quarter(3, 2001), 734.4);
series.add(new Quarter(4, 2001), 453.2);
series.add(new Quarter(1, 2002), 500.2);
series.add(new Quarter(2, 2002), null);
series.add(new Quarter(3, 2002), 734.4);
series.add(new Quarter(4, 2002), 453.2);
```

One or more `TimeSeries` objects can be aggregated to form a dataset for a chart using the `TimeSeriesCollection` class.

29.14.3 Constructors

To create a named time series containing no data:

`public TimeSeries(String name);`
Creates an empty time series for *daily* data (that is, one value per day).

To create a time series for a frequency other than daily, use this constructor:

`public TimeSeries(String name, Class timePeriodClass);`
Creates an empty time series. The caller specifies the time period by specifying the class of the `RegularTimePeriod` subclass (for example, `Month.class`).

The final constructor allows you to specify descriptions for the domain and range of the data:

`public TimeSeries(String name, String domain, String range,`
`Class timePeriodClass);`
Creates an empty time series. The caller specifies the time period, plus strings describing the domain and range.

29.14.4 Attributes

Each instance of `TimeSeries` has the following attributes:

Attribute:	Description:
name	The name of the series (inherited from `Series`).
domain-description	A description of the time period domain (for example, "Quarter"). The default is "Time".
range-description	A description of the value range (for example, "Price"). The default is "Value".

29.14.5 Methods

To find out how many data items are in a series:

`public int getItemCount()`
Returns the number of data items in the series.

To retrieve a particular value from a series by the index of the item:

`public TimeSeriesDataItem getDataPair(int item)`
Returns a data item. The `item` argument is a zero-based index.

To retrieve a particular value from a series by time period:

```
public TimeSeriesDataItem getDataPair(RegularTimePeriod period)
```
Returns the data item (if any) for the specified time period.

To add a value to a time series:

```
public void add(RegularTimePeriod period, Number value)
throws SeriesException;
```
Adds a new value (`null` permitted) to the time series. Throws an exception if the time period is not unique within the series.

You can create a time series that automatically discards "old" data. This is done by specifying a *history-count* attribute:

```
public void setHistoryCount(int count);
```
Sets the *history-count* attribute, which is the number of time periods in the "history" for the time series. When a new data value is added, any data that is more than *history-count* periods old is automatically discarded.

29.14.6 Notes

You can calculate the moving average of a time series using the `MovingAverage` utility class.

The `TimeSeriesDemo` class provides an example of how to create a dataset for a chart using this class.

See Also

`TimePeriod`, `TimeSeriesCollection`.

29.15 TimeSeriesCollection

29.15.1 Overview

A collection of zero, one or many `TimeSeries` objects. A useful feature of this class is that it implements the `XYDataset` and `IntervalXYDataset` interfaces, so it can be used as a dataset for creating charts.

29.15.2 Constructors

To create an *empty* time series collection:

```
public TimeSeriesCollection();
```
Creates a new time series collection, initially empty.

To create a collection containing a single time series (more can be added later):

```
public TimeSeriesCollection(TimeSeries series);
```
Creates a new time series collection, containing a single time series.

Once a collection has been constructed, you are free to add additional time series to the collection.

29.15.3 Methods

To find out how many time series objects are in the collection:

`public int getSeriesCount();`
Returns the number of time series objects in the collection.

To get a reference to a particular series:

`public TimeSeries getSeries(int series);`
Returns a reference to a series in the collection.

To get the name of a series:

`public String getSeriesName(int series);`
Returns the name of a series in the collection. This method is provided for convenience.

To add a series to the collection:

`public void addSeries(TimeSeries series);`
Adds the series to the collection. Registered listeners are notified that the collection has changed.

To get the number of items in a series:

`public int getItemCount(int series);`
Returns the number of items in a series. This method is implemented as a requirement of the `XYDataset` interface.

29.15.4 Notes

Points to note:

- this class extends `AbstractSeriesDataset` to provide some of the basic series information.
- this class implements the `XYDataset` and `IntervalXYDataset` interfaces.

See Also:
`AbstractSeriesDataset`, `BasicTimeSeries`, `XYDataset` and `IntervalXYDataset`.

29.16 TimeSeriesDataItem

29.16.1 Overview

Associates a numerical value with a time period. This class is used by the `TimeSeries` class.

This class has a number of important features:

- the class implements the `Comparable` interface, allowing data items to be sorted into time order using standard Java API calls.
- instances of this class can be easily cloned.
- the time period element is immutable, so that when a collection of objects is held in sorted order, the sorted property cannot inadvertently be broken.

29.17 Week

29.17.1 Overview

A subclass of `RegularTimePeriod` that represents one week in a particular year. This class is designed to be used with the `TimeSeries` class, but (hopefully) is general enough to be used in other situations.

29.17.2 Constructors

To construct a `Week` instance:

`public Week(int week, Year year);`
Creates a new `Week` instance. The `week` argument should be in the range 1 to 52.

`public Week(int week, int year);`
Creates a new `Week` instance.

To construct a `Week` instance based on a `java.util.Date`:

`public Week(Date time);`
Creates a new `Week` instance.

A default constructor is provided:

`public Week();`
Creates a new `Week` instance based on the current system time.

29.17.3 Methods

To access the week:

`public int getWeek();`
Returns the week (in the range 1 to 52).

To access the year:

`public Year getYear();`
Returns the year.

There is no method to *set* the week or the year, because this class is immutable.

Given a `Week` object, you can create an instance representing the previous week or the next week:

`public TimePeriod previous();`
Returns the previous week, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next week, or null if the upper limit of the range is reached.

To convert a `Week` object to a `String` object:

`public String toString();`
Returns a string representing the week.

29.17.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Week` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`Year`.

29.18 Year

29.18.1 Overview

A calendar year—this class extends `RegularTimePeriod`.

One use for this class is in the construction of `TimeSeries` objects for *annual data*.

29.18.2 Constructors

To create a new year:

```
public Year(int year);
```
Creates a new `Year` instance. The `year` argument should be in the range 1900 to 9999.

To construct a `Year` instance based on a `java.util.Date`:

```
public Year(Date time);
```
Creates a new `Year` instance.

A default constructor is provided:

```
public Year();
```
Creates a new `Year` instance based on the current system time.

29.18.3 Methods

To access the year:

```
public int getYear();
```
Returns the year.

There is no method to *set* the year, because this class is immutable.

Given a `Year` object, you can create an instance representing the previous year:

```
public RegularTimePeriod previous();
```
Returns the previous year, or `null` if the lower limit of the range is reached.

...or the next:

```
public RegularTimePeriod next();
```
Returns the next year, or `null` if the upper limit of the range is reached.

To convert a `Year` object to a `String` object:

`public String toString();`
Returns a string representing the year.

To convert a `String` object to a `Year` object:

`public static Year parseYear(String s) throws TimePeriodFormatException;`
Parses the string and, if possible, returns a `Year` object.

29.18.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Year` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

30 Package: org.jfree.data.XML

30.1 Introduction

This package contains interfaces and classes for reading datasets from XML.

30.2 CategoryDatasetHandler

30.2.1 Overview

Not yet documented.

30.3 CategorySeriesHandler

30.3.1 Overview

Not yet documented.

30.4 DatasetReader

30.4.1 Overview

Not yet documented.

30.5 DatasetTags

30.5.1 Overview

Not yet documented.

30.6 ItemHandler

30.6.1 Overview

Not yet documented.

30.7 KeyHandler

30.7.1 Overview

Not yet documented.

30.8 PieDatasetHandler

30.8.1 Overview

Not yet documented.

30.9 RootHandler

30.9.1 Overview

Not yet documented.

30.10 ValueHandler

30.10.1 Overview

Not yet documented.

A The GNU Lesser General Public Licence

A.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public Licence (LGPL). The full text of this licence is reproduced in this appendix. You should read and understand this licence before using JFreeChart in your own projects.

If you are not familiar with the idea of *free software*, you can find out more at the Free Software Foundation's web site:

`http://www.fsf.org`

Please send e-mail to `david.gilbert@object-refinery.com` if you have any questions about the licensing of JFreeChart (but please read section A.3 first).

A.2 The Licence

The following licence has been used for the distribution of the JFreeChart class library:

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

* a) The modified work must itself be a software library.

* b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

* c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

* d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must

be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

* a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

* b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

* c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

* d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

* e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a

special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

* b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work. 8. You may not copy, modify, sublicense, link with, or distribute the Library except

as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In

such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for
more details.

You should have received a copy of the GNU Lesser General Public License
```

```
along with this library; if not, write to the Free Software Foundation,
Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the library
'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

A.3 Frequently Asked Questions

A.3.1 Introduction

Some of the most frequently asked questions about JFreeChart concern the licence. I've published this FAQ to help developers understand my choice of licence for JFreeChart. If anything is unclear, or technically incorrect, please e-mail me (david.gilbert@object-refinery.com) and I will try to improve the text.

A.3.2 Questions and Answers

1. "Can I incorporate JFreeChart into a proprietary (closed-source) application?"

Yes, the GNU Lesser General Public Licence (LGPL) is specifically designed to allow this.

2. "Do I have to pay a licence fee to use JFreeChart?"

No, JFreeChart is free software. You are not required to pay a fee to use JFreeChart. All that we ask is that you comply with the terms of the licence, which (for most developers) is not very difficult.

If you want to make a financial contribution to the JFreeChart project, you can buy one or more copies of the documentation. This is appreciated, but not required.

3. "If I use JFreeChart, do I have to release the source code for my application under the terms of the LGPL?"

No, you can choose whatever licence you wish for your software. But when you distribute your application, you must include the complete source code for JFreeChart—including any changes you make to it—under the terms of the LGPL. Your users end up with the same rights in relation to JFreeChart as you have been granted under the LGPL.

4. "My users will never look at the source code, and if they did, they wouldn't know what to do with it...why do I have to give it to them?"

The important point is that your users have access to the source code—whether or not they choose to use it is up to them. Bear in mind that non-technical users *can* make use of the source code by hiring someone else to work on it for them.

5. "What are the steps I must follow to release software that incorporates JFreeChart?"

The steps are listed in the licence (see section 6 especially). The most important things are:

- include a notice in your software that it uses the JFreeChart class library, and that the library is covered by the LGPL;
- include a copy of the LGPL so your users understand that JFreeChart is distributed WITHOUT WARRANTY, and the rights that they have under the licence;
- include the complete source code for the version of the library that you are distributing (or a written offer to supply it on demand);

6. "I want to display the JFreeChart copyright notice, what form should it take?"

Try this:

> *This software incorporates JFreeChart, (C)opyright 2000-2003 by Simba Management Limited and Contributors.*

7. "The LGPL is unnecessarily complicated!"

OK, that's not a question, but the point has been raised by a few developers.

Yes, the LGPL is complicated, but only out of necessity. The complexity is mostly related to the difficulty of defining (in precise legal terms) the relationship between a free software library and a proprietary application that uses the library.

A useful first step towards understanding the LGPL is to read the GNU General Public Licence (GPL). It is a much simpler licence, because it does not allow free software to be combined with non-free (or proprietary) software. The LGPL is a superset of the GPL (you are free to switch from the LGPL to the GPL at any time), but slightly more "relaxed" in that it allows you to combine free and non-free software.

A final note, some of the terminology in the LGPL is easier to understand if you keep in mind that the licence was originally developed with statically-linked C programs in mind. Ensuring that it is possible to relink a modified free library with a non-free application, adds significant complexity to the licence. For Java libraries, where code is dynamically linked, modifying and rebuilding a free library for use with a non-free application needn't be such a big issue, particularly if the free library resides in its own jar file.

8. "Who developed the licence?"

The licence was developed by the Free Software Foundation and has been adopted by many thousands of free software projects. You can find out more information at the Free Software Foundation website:

`http://www.fsf.org`

The Free Software Foundation performs important work, please consider supporting them financially.

9. "Have you considered releasing JFreeChart under a different licence, such as an "Apache-style" licence?"

Yes, a range of licences was considered for JFreeChart, but now that the choice has been made there are no plans to change the licence in the future.

A publication by Bruce Perens was especially helpful in comparing the available licences:

`http://www.oreilly.com/catalog/opensources/book/perens.html`

In the end, the LGPL was chosen because it is the closest fit in terms of my goals for JFreeChart. It is not a perfect licence, but there is nothing else that comes close (except the GPL) in terms of protecting the freedom of JFreeChart for everyone to use. Also, the LGPL is very widely used, and many developers are already familiar with its requirements.

Some other open source licences (for example the Apache Software Licence) allow open source software to be packaged and redistributed without source code. These licences offer more convenience to developers (especially in large companies) than the LGPL, but they allow a path from open source software to closed source software, which is not something I want to allow for JFreeChart.