The JFreeChart Class Library

Version 0.9.18

Developer Guide

Written by David Gilbert

April 28, 2004

IMPORTANT NOTICE:
If you choose to use this document you do so entirely at your own risk.

Contents

Chapter 1

Introduction

1.1 What is JFreeChart?

1.1.1 Overview

JFreeChart is a free chart library for the Java(tm) platform. It is designed for use in applications, applets, servlets and JSP. JFreeChart is distributed with

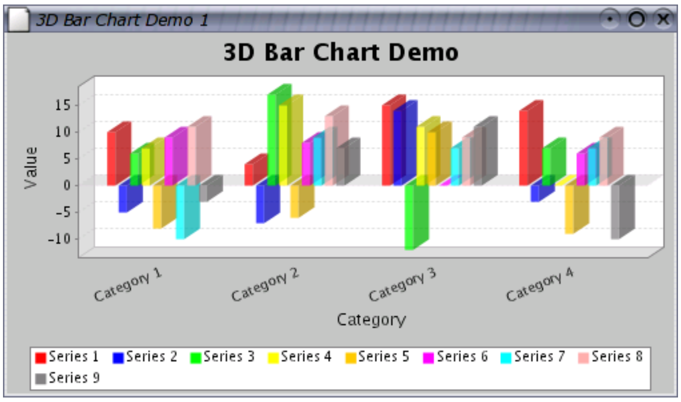


Figure 1.1: A sample chart

complete source code subject to the terms of the GNU Lesser General Public Licence (see Appendix B for details).

1.1.2 Features

JFreeChart can generate pie charts, bar charts (regular and stacked, with an 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:

- data is accessible from any implementation of the defined interfaces;
- export to PNG and JPEG;
- export to any format with a `Graphics2D` implementation including:
  - PDF via iText (`http://www.lowagie.com/iText/`);
  - SVG via Batik (`http://xml.apache.org/batik/`);
- tool tips;
- interactive zooming;
- chart mouse events;
- annotations;
- HTML image map generation;
- works in applications, servlets, JSP (thanks to the Cewolf project[1]) and applets;
- 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 (JDK 1.2.2 or later).

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 discussion forum and more.

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

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 and running the demos.
- a premium version, the "JFreeChart Developer Guide", is available only to those that have paid for it, and includes additional tutorial chapters and reference documentation for the JFreeChart classes.

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, Daniel Bridenbecker, Nicolas Brodu, David Browning, 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, Arnaud Lelievre, David Li, Tin Luu, Craig MacFarlane, Achilleus Mantzios, Thomas Meier, Aaron Metzger, Jim Moore, Jonathan Nash, Barak Naveh, David M. O'Donnell, Krzysztof Paz, Tomer Peretz, Andrzej Porebski, Luke Quinane, Viktor Rajewski, Eduardo Ramalho, Michael Rauch, Cameron Riley, Dan Rivett, Michel Santos, Thierry Saura, Andreas Schneider, Jean-Luc Schwab, Bryan Scott, Roger Studner, Irv Thomae, Eric Thomas, Rich Unger, Daniel van Enckevort, Laurence Vanhelsuwé, Sylvain Vieujot, Jelai Wang, 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`

Chapter 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. For other examples, please try the demo applications included in the JFreeChart distribution (source code is included in the `src/org/jfree/chart/demo` directory).

2.2 Pie Charts

JFreeChart can create *pie charts* using any data that conforms to the `PieDataset` interface. Figure 2.1 shows a simple pie chart.

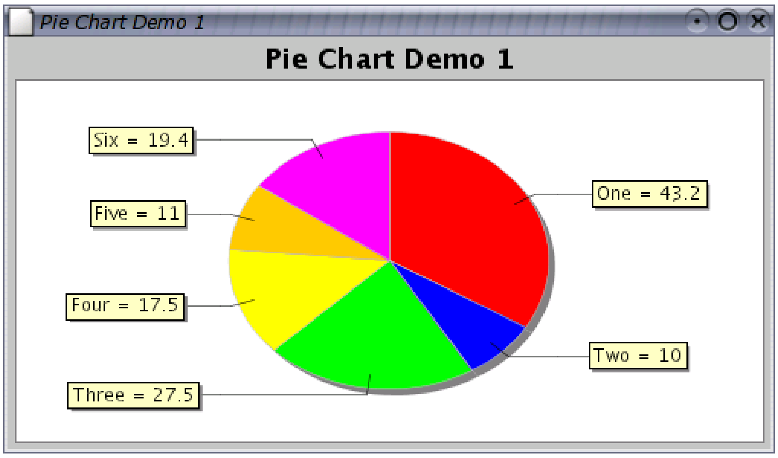


Figure 2.1: A simple pie chart

Individual pie sections can be "exploded", as shown in figure 2.2.
You can also display pie charts with a 3D effect, as shown in figure 2.3.
At the current time it is *not* possible to explode sections of the 3D pie chart.

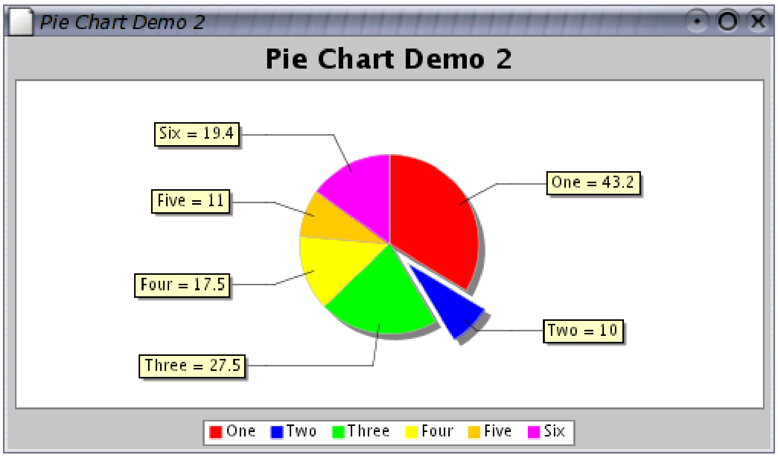


Figure 2.2: A pie chart with an "exploded" section

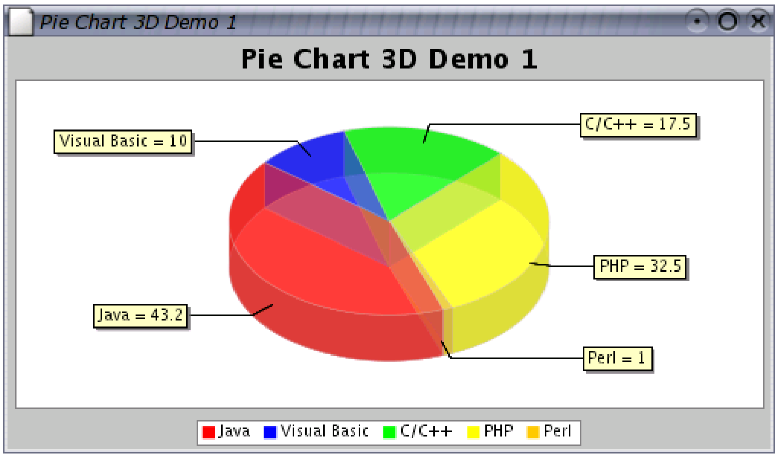


Figure 2.3: A pie chart drawn with a 3D effect

2.3 Bar Charts

A range of bar charts can be created with JFreeChart, using any data. that conforms to the `CategoryDataset` interface. Figure 2.4 shows a bar chart with a vertical orientation.

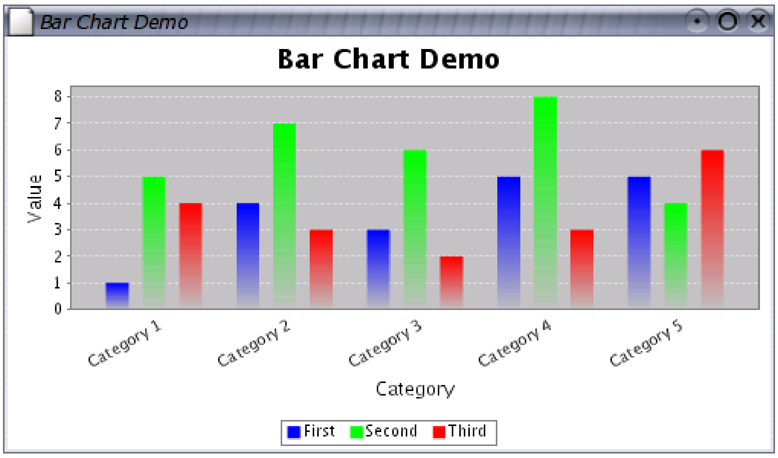


Figure 2.4: A vertical bar chart

Bar charts can be displayed with a 3D effect as shown in figure 2.5.

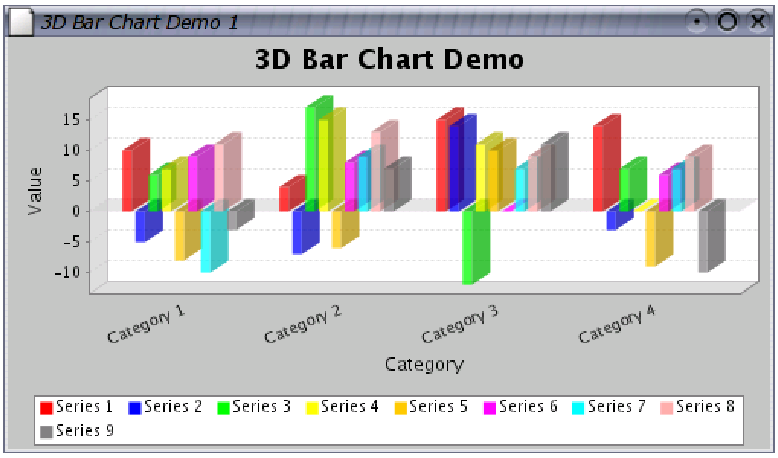


Figure 2.5: A bar chart with 3D effect

Another variation, the *waterfall chart*, is shown in figure 2.6.

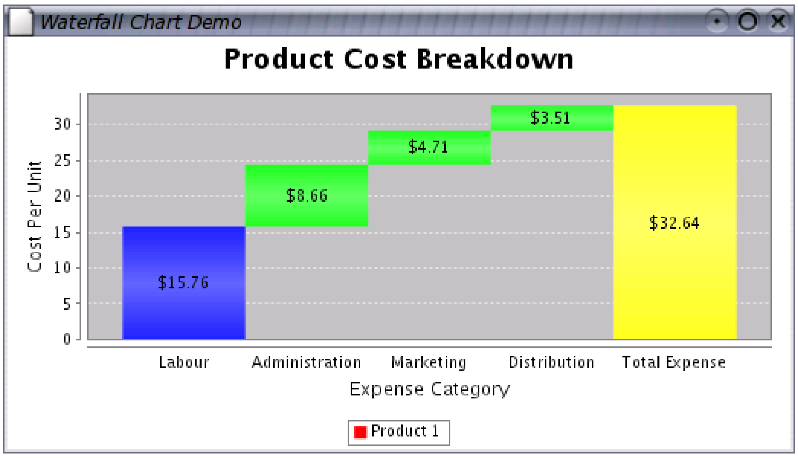


Figure 2.6: A waterfall chart

2.4 Line Chart

The *line chart* can be generated using the same `CategoryDataset` that is used for the bar charts—figure 2.7 shows an example.

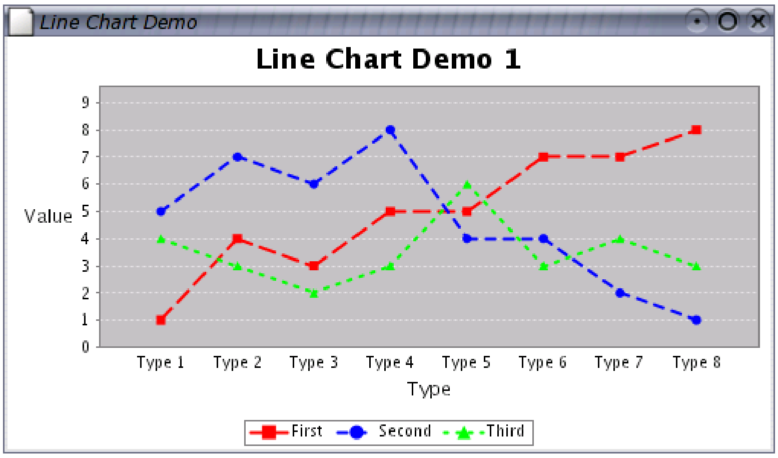


Figure 2.7: A line chart

2.5 XY Plots

A third type of dataset, the `XYDataset`, is used to generate a range of chart types.

The standard *XY plot* has numerical x and y axes. By default, lines are drawn between each data point—see figure 2.8.

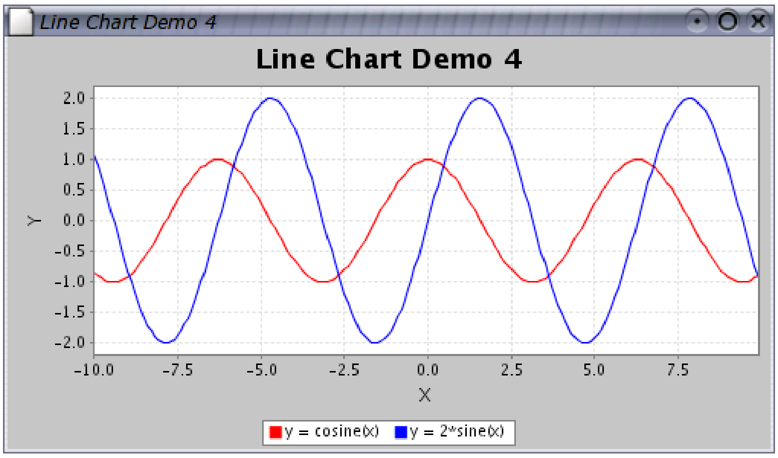


Figure 2.8: A line chart

Scatter plots can be drawn by drawing a shape at each data point, rather than connecting the points with lines—an example is shown in figure 2.9.

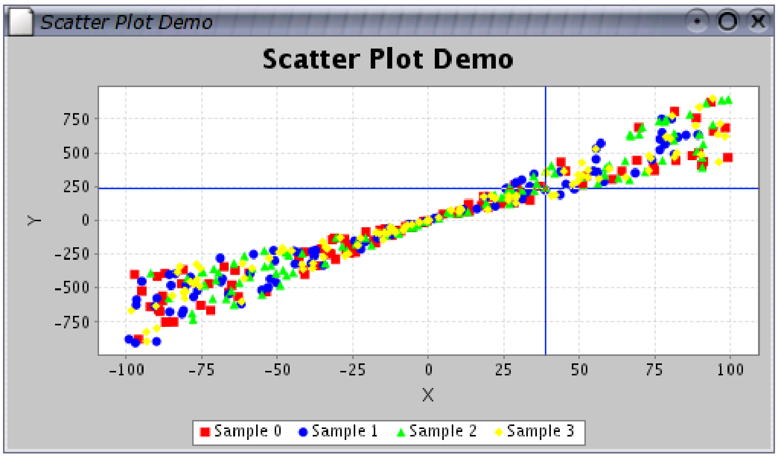


Figure 2.9: A scatter plot

2.6 Time Series Charts

JFreeChart supports *time series charts*, as shown in figure 2.10.

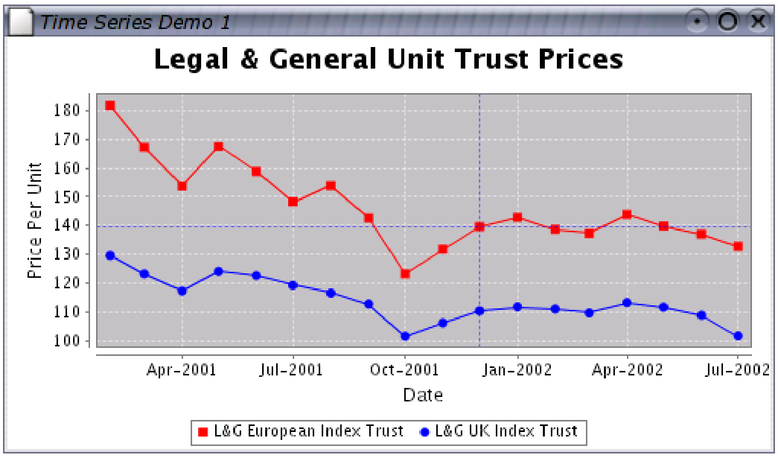


Figure 2.10: A time series chart

It is straightforward to add a moving average line to a time series chart—see figure 2.11 for an example.

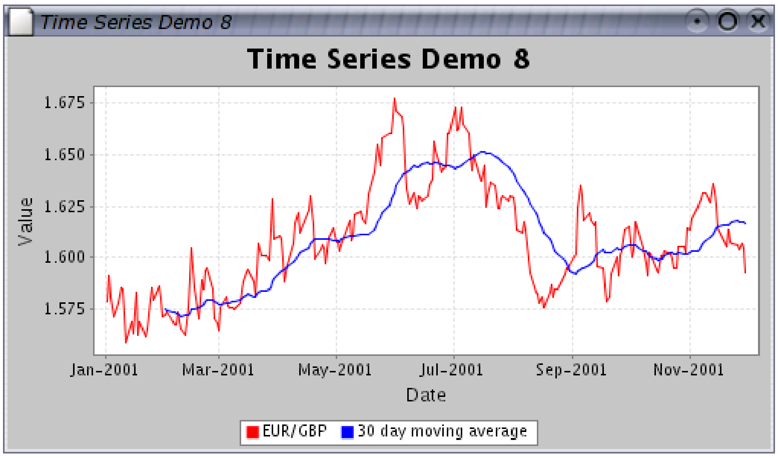


Figure 2.11: A time series chart with a moving average

Using a `HighLowDataset` (an extension of `XYDataset`) you can display *high-low-open-close* data, see figure 2.12 for an example.

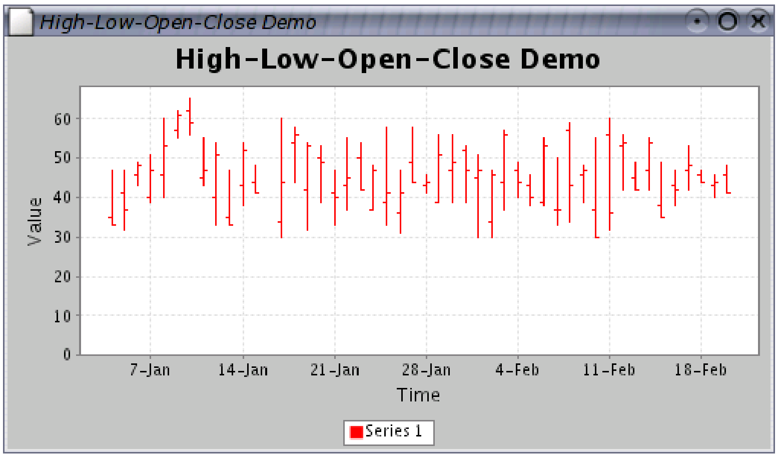


Figure 2.12: A high-low-open-close chart

2.7 Histograms

Histograms can be generated using an `IntervalXYDataset` (another extension of `XYDataset`), see figure 2.13 for an example.

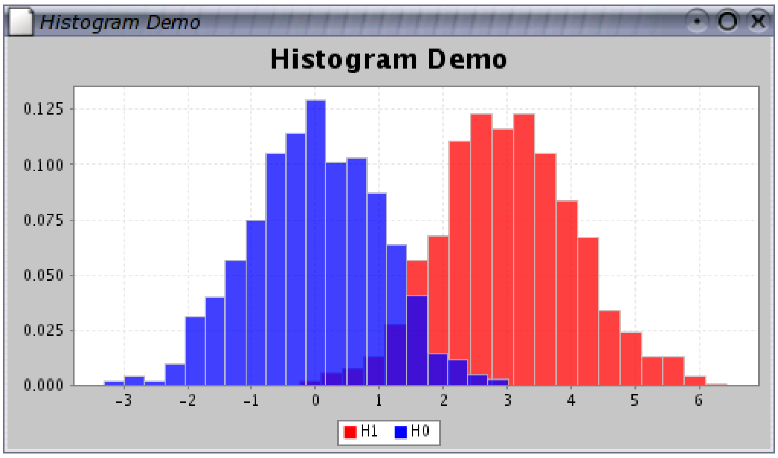


Figure 2.13: A histogram

2.8 Area Charts

You can generate an *area chart* for data in a `CategoryDataset` or an `XYDataset`. Figure 2.14 shows an example.

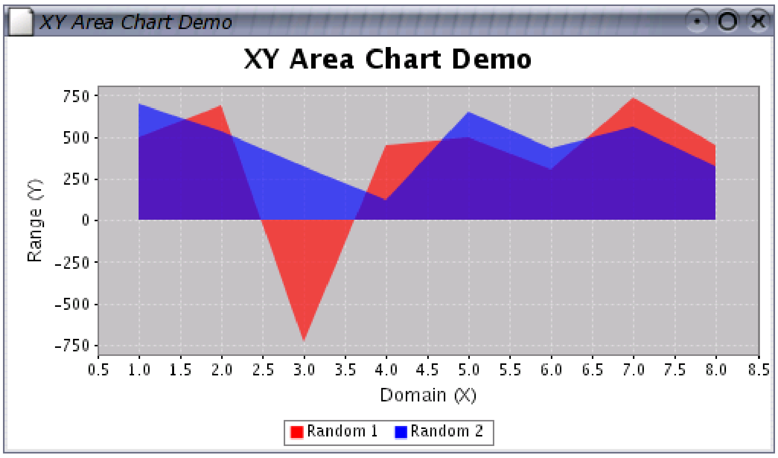


Figure 2.14: An area chart

JFreeChart also supports the creation of *stacked area charts* as shown in figure 2.15.

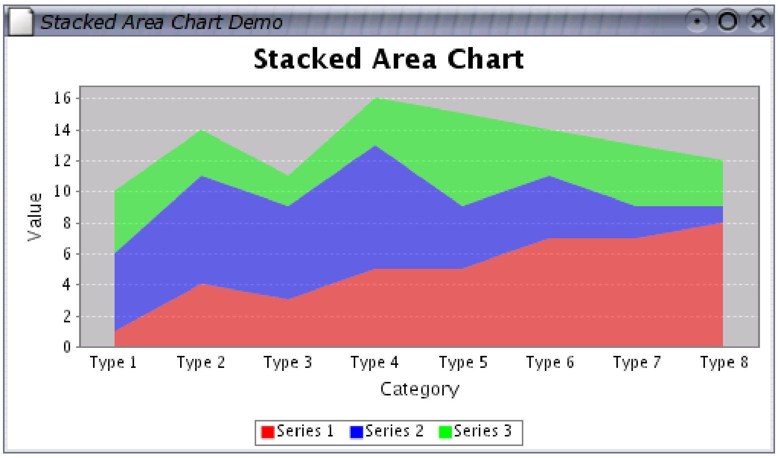


Figure 2.15: A stacked area chart

2.9 Difference Chart

A *difference chart* highlights the difference between two series (see figure 2.16). A second example, shown in figure 2.17 shows how a date axis can be used for the range values.

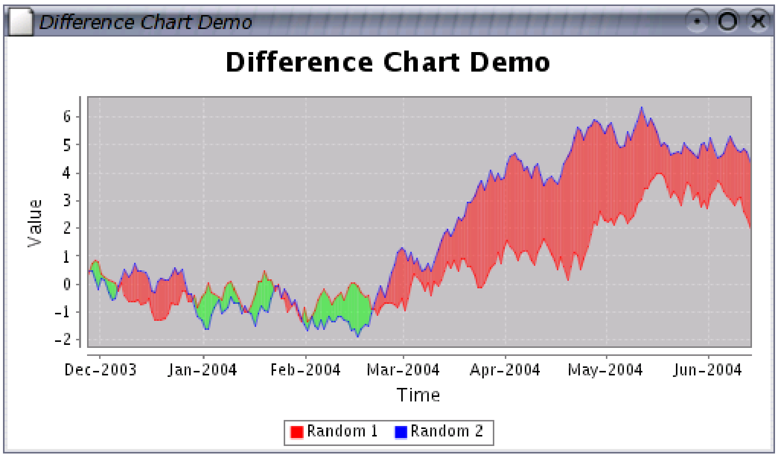


Figure 2.16: A difference chart

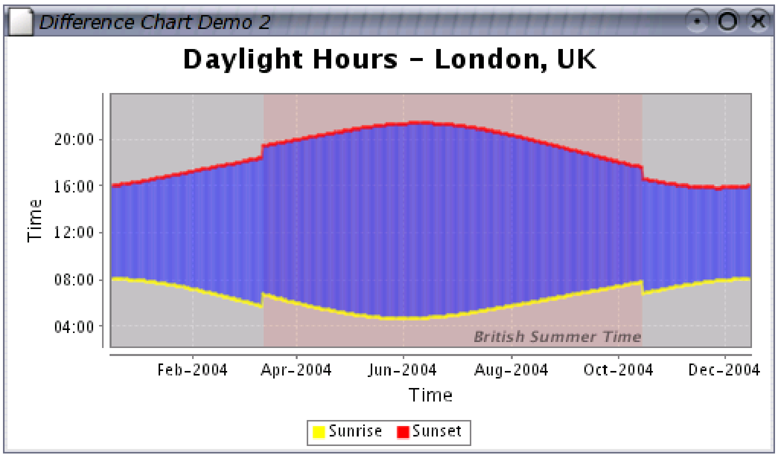


Figure 2.17: A difference chart with times on the range axis

2.10 Step Chart

A *step chart* displays numerical data as a sequence of "steps"—an example is shown in figure 2.18.

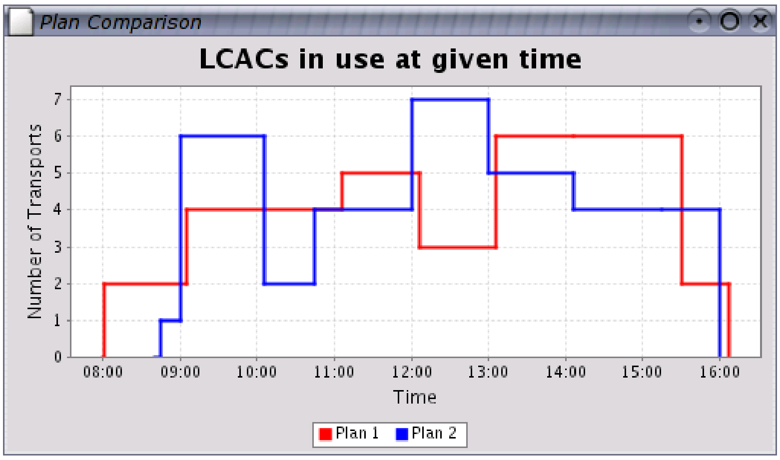


Figure 2.18: A step chart

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

2.11 Gantt Chart

Gantt charts can be generated using data from an `IntervalCategoryDataset`, as shown in figure 2.19.

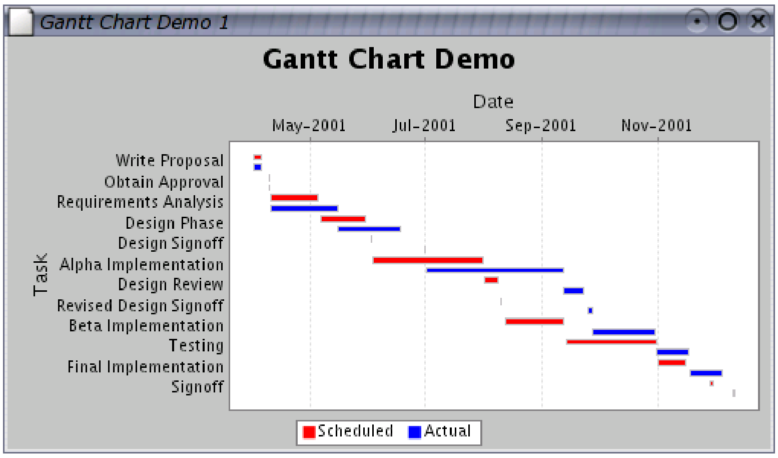


Figure 2.19: A Gantt chart

Another example, showing subtasks and progress indicators, is shown in figure 2.20.

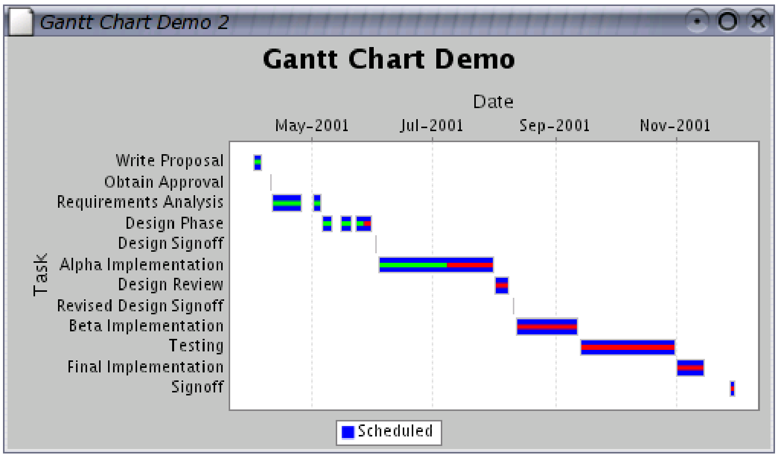


Figure 2.20: A Gantt chart with progress indicators

2.12 Multiple Axis Charts

JFreeChart has support for charts with multiple axes. Figure 2.21 shows a *price-volume chart* that demonstrates this feature.

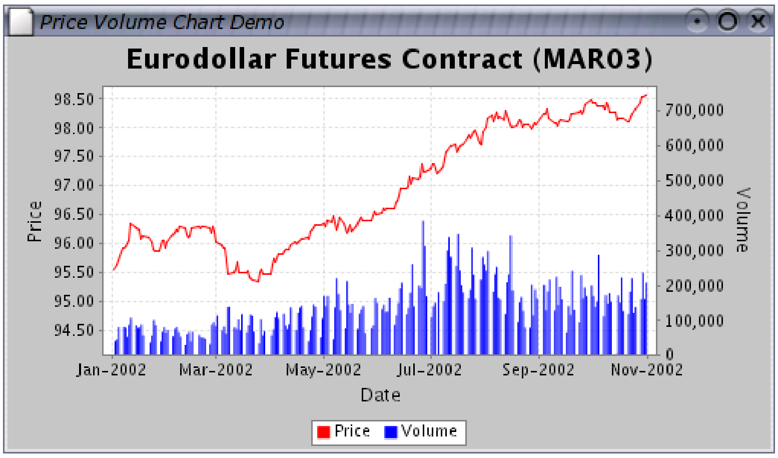


Figure 2.21: A price-volume chart

This feature is supported by the `CategoryPlot` and `XYPlot` classes. Figure 2.22 shows an example with four range axes.

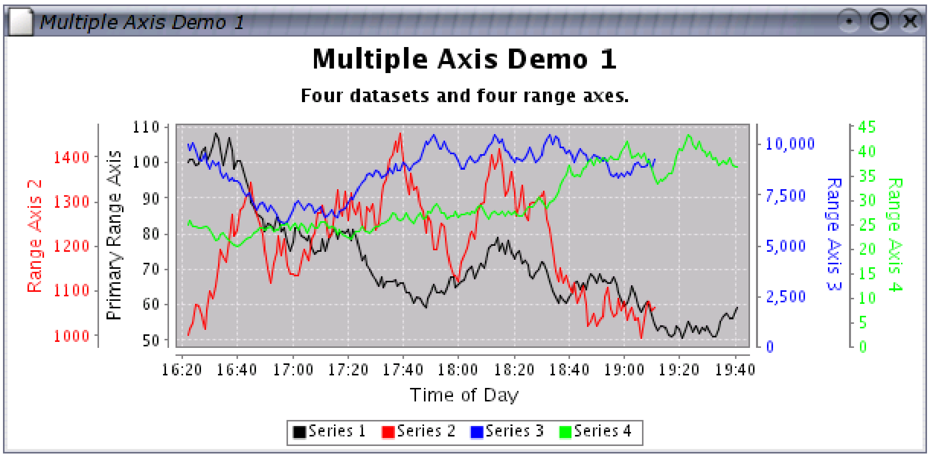


Figure 2.22: A chart with multiple axes

2.13 Combined and Overlaid Charts

JFreeChart supports combined and overlaid charts. Figure 2.23 shows a line chart overlaid on top of a bar chart.

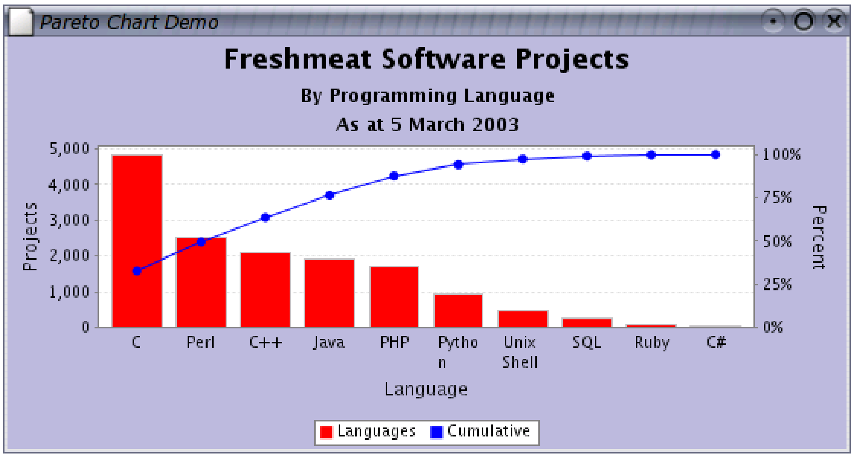


Figure 2.23: An overlaid chart

It is possible to combine several charts that share a common domain axis, as shown in figure 2.24.

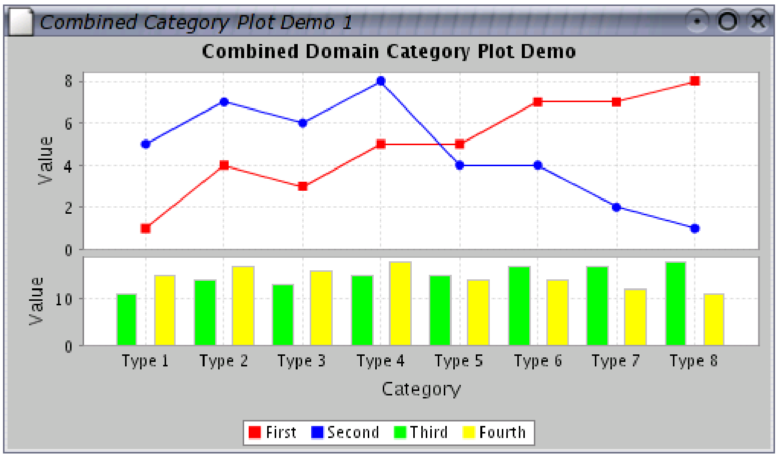


Figure 2.24: A chart with a combined domain

In a similar way, JFreeChart can combine several charts that share a common range axis, see figure 2.25.

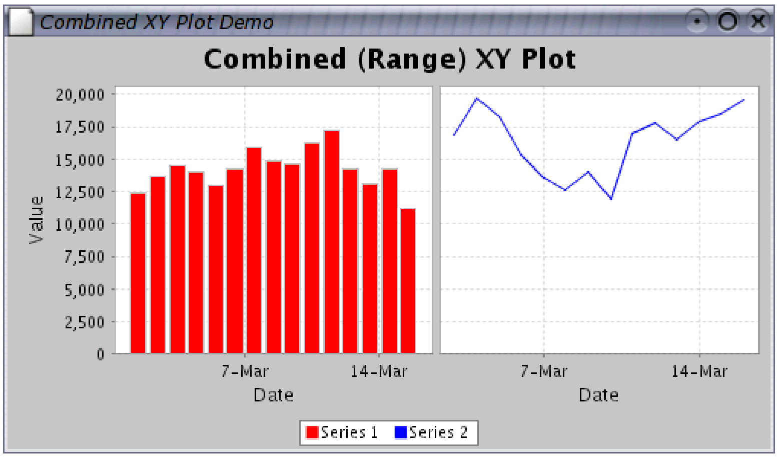


Figure 2.25: A chart with a combined range

2.14 Future Development

JFreeChart is *free software*,[1] so anyone can extend it and add new features to it. Already, more than 50 developers from around the world have contributed 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...

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

Chapter 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-0.9.18.tar.gz`	JFreeChart for Linux/Unix.
`jfreechart-0.9.18.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 `0.9.3`). 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-0.9.18`) will be created in the same location as the `zip` or `tar.gz` archive 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-0.9.18.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.18`.

3.3.2 Unpacking on Windows

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

```
jar -xvf jfreechart-0.9.18.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.18`.

3.3.3 The Files

The top-level directory (`jfreechart-0.9.18`) contains the files and directories listed in the following table:

File/Directory:	**Description:**
`ant`	A directory containing an Ant `build.xml` script. You can use this script to rebuild JFreeChart from the source code included in the distribution.
`CHANGELOG.txt`	A log of changes made to JFreeChart since the previous release.
`checkstyle`	A directory containing several Checkstyle property files. These define the coding conventions used in the JFreeChart source code.
`jfreechart-0.9.18.jar`	The JFreeChart runtime jar file.
`jfreechart-0.9.18-demo.jar`	A runnable 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.txt`	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.txt` 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-0.9.18-demo.jar
```

Alternatively, you can specify the classpath manually:

```
java -classpath lib/jcommon-0.9.3.jar:jfreechart-0.9.18.jar:
lib/log4j-1.2.8.jar:jfreechart-0.9.18-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-0.9.18` directory.

Chapter 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 the pie chart shown in figure **??**. 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 dataset = new DefaultPieDataset();
dataset.setValue("Category 1", 43.2);
dataset.setValue("Category 2", 27.9);
dataset.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)

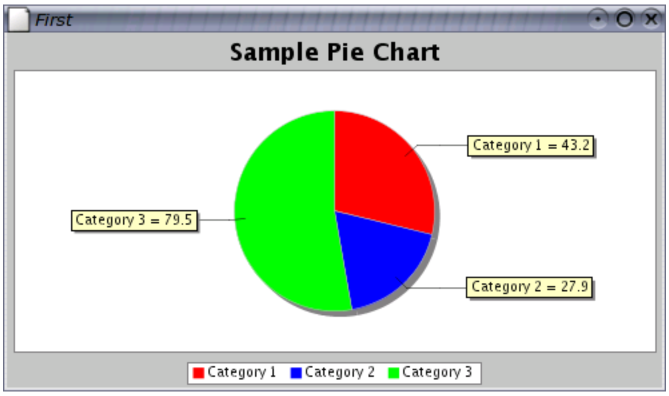


Figure 4.1: A pie chart created using ***First.java***

provides a convenient implementation of this interface, but you are free to develop an alternative dataset implementation if you want to.[1]

4.2.3 Creating a Pie Chart

Step two concerns how we will present the dataset created in the previous section. 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",
    dataset,
    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:

[1] 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.

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

        // create a chart...
        JFreeChart chart = ChartFactory.createPieChart(
            "Sample Pie Chart",
            dataset,
            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...

Chapter 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 obtained via the `CategoryDataset` interface (it is also possible to use the `IntervalXYDataset` interface, but more on that later).

5.2 A Bar Chart

5.2.1 Overview

A *bar chart* is created using data from a `CategoryDataset`, and represents each data item as a bar where the length of the bar is equal to the data value. This section presents a sample application that generates the chart shown in figure 5.1.

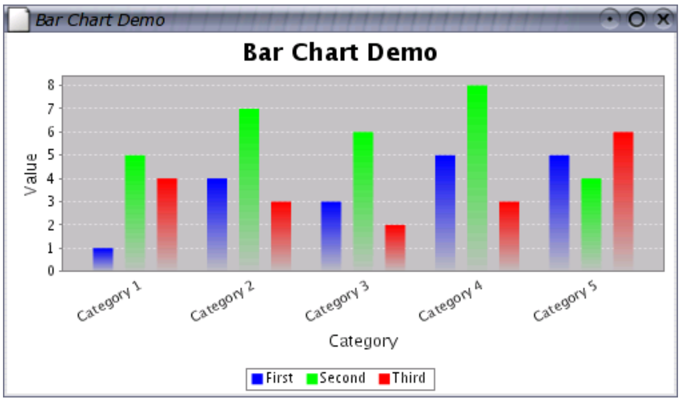


Figure 5.1: A sample bar chart

The full source code (`BarChartDemo.java`) is included in the JFreeChart distribution, in the `src/org/jfree/chart/demo` directory.

5.2.2 The Dataset

The first step in generating the chart is to create a dataset. You can use *any class* that implements the `CategoryDataset` interface—for the example, we have used the `DefaultCategoryDataset` class (included in the JFreeChart distribution):

```
/**
 * Returns a sample dataset.
 *
 * @return The dataset.
 */
private CategoryDataset createDataset() {

    // 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";

    // 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(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, 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);

    return dataset;

}
```

Notice that we have used `String` objects as the row and column keys for the data values. You can use *any class* that implements the `Comparable` interface as the keys for your data values.

5.2.3 Constructing the Chart

The `createBarChart()` method in the `ChartFactory` class provides a convenient way to create the chart:[1]

```
// create the chart...
JFreeChart chart = ChartFactory.createBarChart(
    "Bar Chart Demo",          // chart title
    "Category",                // domain axis label
    "Value",                   // range axis label
```

[1] Take a look at the source code for this method, if you are interested to know how the bar chart is constructed from the components (axes, plots, renderers etc.) in the JFreeChart library.

```
            dataset,                  // data
            PlotOrientation.VERTICAL,
            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.

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 "auto tick units" on the range axis (so that the tick labels always display integer values);
- gradient paint is used for the series colors;

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 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());
```

The bar renderer is modified so that bar outlines are not drawn, and GradientPaint instances are used for the series colors:

```
// disable bar outlines...
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setDrawBarOutline(false);

// set up gradient paints for series...
GradientPaint gp0 = new GradientPaint(
    0.0f, 0.0f, Color.blue,
    0.0f, 0.0f, Color.lightGray
);
GradientPaint gp1 = new GradientPaint(
    0.0f, 0.0f, Color.green,
    0.0f, 0.0f, Color.lightGray
);
GradientPaint gp2 = new GradientPaint(
    0.0f, 0.0f, Color.red,
    0.0f, 0.0f, Color.lightGray
);
renderer.setSeriesPaint(0, gp0);
renderer.setSeriesPaint(1, gp1);
renderer.setSeriesPaint(2, gp2);
```

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 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 java.awt.GradientPaint;

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.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.BarRenderer;
import org.jfree.data.CategoryDataset;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demonstration application showing how to create a bar chart.
 *
 * @author David Gilbert
 */
public class BarChartDemo extends ApplicationFrame {

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

        super(title);

        CategoryDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);

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

    }

    /**
     * Returns a sample dataset.
     *
     * @return The dataset.
     */
    private CategoryDataset createDataset() {

        // 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";

        // 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(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, 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);

        return dataset;

    }

    /**
     * Creates a sample chart.
     *
     * @param dataset  the dataset.
     *
     * @return The chart.
     */
    private JFreeChart createChart(CategoryDataset dataset) {

        // create the chart...
        JFreeChart chart = ChartFactory.createBarChart(
            "Bar Chart Demo",         // chart title
            "Category",               // domain axis label
            "Value",                  // range axis label
            dataset,                  // data
            PlotOrientation.VERTICAL,
            true,                     // include legend
            true,                     // tooltips?
            false                     // URLs?
        );

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

        // 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();

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

        // disable bar outlines...
        BarRenderer renderer = (BarRenderer) plot.getRenderer();
        renderer.setDrawBarOutline(false);

        // set up gradient paints for series...
        GradientPaint gp0 = new GradientPaint(
            0.0f, 0.0f, Color.blue,
            0.0f, 0.0f, Color.lightGray
        );
        GradientPaint gp1 = new GradientPaint(
            0.0f, 0.0f, Color.green,
            0.0f, 0.0f, Color.lightGray
        );
        GradientPaint gp2 = new GradientPaint(
            0.0f, 0.0f, Color.red,
            0.0f, 0.0f, Color.lightGray
        );
        renderer.setSeriesPaint(0, gp0);
        renderer.setSeriesPaint(1, gp1);
        renderer.setSeriesPaint(2, gp2);
```

```
        // OPTIONAL CUSTOMISATION COMPLETED.

        return chart;

    }

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

        BarChartDemo demo = new BarChartDemo("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 = chart.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 = chart.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 *N x (M - 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.

Chapter 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 shown in figure 6.1.

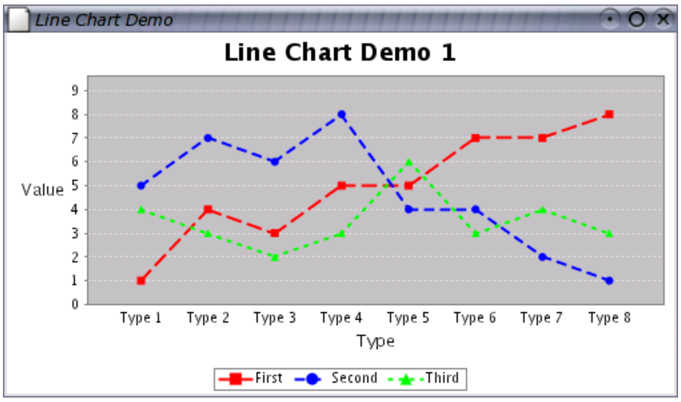


Figure 6.1: A sample line chart

The full source code (`LineChartDemo1.java`) is included in the JFreeChart distribution, in the `src/org/jfree/chart/demo` directory.

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:

```
/**
 * Creates a sample dataset.
 *
 * @return The dataset.
 */
private CategoryDataset createDataset() {

    // 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);

    return dataset;

}
```

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:

```
// create the chart...
```

```
JFreeChart chart = ChartFactory.createLineChart(
    "Line Chart Demo 1",       // chart title
    "Type",                    // domain axis label
    "Value",                   // range axis label
    dataset,                   // data
    PlotOrientation.VERTICAL,  // orientation
    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(new Color(0xCC, 0xCC, 0xFF));
```

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 draw shapes (as well as lines), customise the line stroke used for each series, and display labels for each data item:

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

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

```
        )
    );
    renderer.setSeriesStroke(
        2,
        new BasicStroke(
            2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
            1.0f, new float[] {2.0f, 6.0f}, 0.0f
        )
    );

    renderer.setItemLabelsVisible(true);
```

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 distribution.

```
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.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.ItemLabelPosition;
import org.jfree.chart.renderer.LineAndShapeRenderer;
import org.jfree.data.CategoryDataset;
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);

        CategoryDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);

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

    }

    private CategoryDataset createDataset() {

        // 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);

        return dataset;

    }

    private JFreeChart createChart(CategoryDataset dataset) {

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

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

        chart.setBackgroundPaint(new Color(0xCC, 0xCC, 0xFF));

        CategoryPlot plot = chart.getCategoryPlot();

        // customise the range axis...
        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
        rangeAxis.setAutoRangeIncludesZero(true);
        rangeAxis.setUpperMargin(0.20);
        rangeAxis.setLabelAngle(Math.PI / 2.0);
```

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

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

        renderer.setItemLabelsVisible(true);
        renderer.setPositiveItemLabelPosition(new ItemLabelPosition());
        renderer.setNegativeItemLabelPosition(new ItemLabelPosition());
        // OPTIONAL CUSTOMISATION COMPLETED.

        return chart;
    }

    public static void main(String[] args) {

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

    }

}
```

6.2.6 A Line Chart Based On An XYDataset

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 chart shown in figure 6.2.

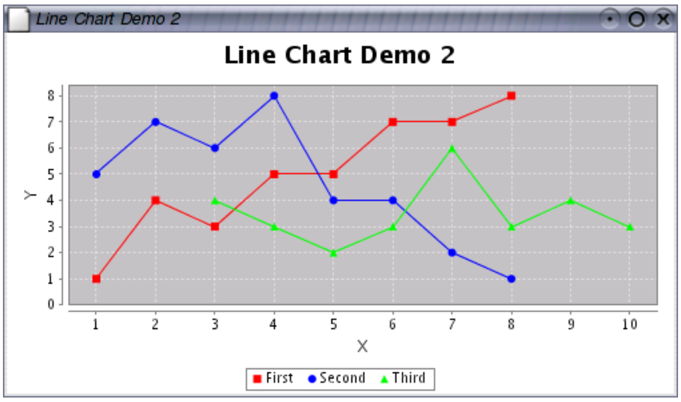


Figure 6.2: A sample line chart using an `XYPlot`

The complete source code (`LineChartDemo2.java`) is included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

The Dataset

For this chart, an XYSeriesCollection is used as the dataset (you can use any implementation of the XYDataset interface). For the purposes of the self-contained demo, we create this dataset in code, as follows:

```
// 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. The dataset will also accept `null` in place of a y-value. When a `null` value is encountered, no connecting line is drawn, resulting in a discontinuous line for the series.

Constructing the Chart

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

```
JFreeChart chart = ChartFactory.createXYLineChart(
    "Line Chart Demo 2",      // chart title
    "X",                      // x axis label
    "Y",                      // y axis label
    dataset,                  // data
    PlotOrientation.VERTICAL,
    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. The chart is created with a legend, and tooltips are enabled (URLs are disabled—these are only used in the creation of HTML image maps).

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 plot background color;
- the legend is configured to draw shapes;
- the axis offsets;
- the color of the domain and range gridlines;
- the renderer is modified to draw shapes as well as lines;
- the tick unit collection for the range 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.white);
```

To get the legend to display the shapes that are associated with each series, we first obtain a reference to the legend, and then change the appropriate flag:

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

Changing the plot background color, the axis offsets, and the color of the gridlines, requires a reference to the plot:

```
// get a reference to the plot for further customisation...
XYPlot plot = chart.getXYPlot();
plot.setBackgroundPaint(Color.lightGray);
plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
plot.setDomainGridlinePaint(Color.white);
plot.setRangeGridlinePaint(Color.white);
```

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

```
StandardXYItemRenderer renderer = (StandardXYItemRenderer) plot.getRenderer();
renderer.setPlotShapes(true);
renderer.setShapesFilled(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`.

The Complete Program

The code for the demonstration application is presented here 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.Spacer;
import org.jfree.chart.StandardLegend;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.StandardXYItemRenderer;
import org.jfree.data.XYDataset;
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 {

    public LineChartDemo2(String title) {

        super(title);

        XYDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
```

```
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        setContentPane(chartPanel);

    }

    private XYDataset createDataset() {

        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);

        return dataset;

    }

    private JFreeChart createChart(XYDataset dataset) {

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

        // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
        chart.setBackgroundPaint(Color.white);

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

        // get a reference to the plot for further customisation...
        XYPlot plot = chart.getXYPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);

        StandardXYItemRenderer renderer = (StandardXYItemRenderer) plot.getRenderer();
```

```
        renderer.setPlotShapes(true);
        renderer.setShapesFilled(true);

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

        return chart;

    }

    public static void main(String[] args) {

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

    }

}
```

Chapter 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 chart shown in figure 7.1.

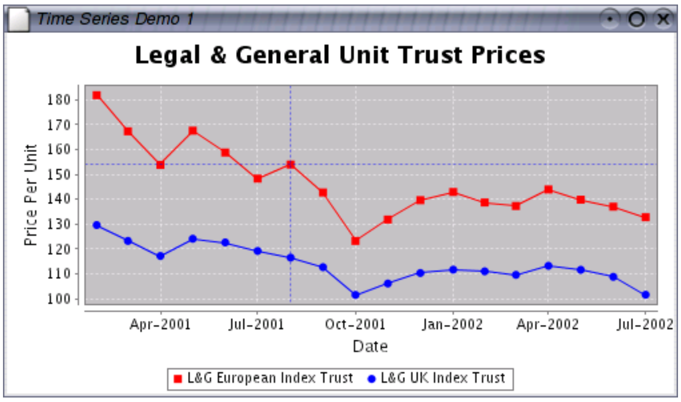


Figure 7.1: A time series chart

The complete source code (`TimeSeriesDemo.java`) 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.setShapesFilled(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 distribution.

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

import java.awt.Color;
import java.text.SimpleDateFormat;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.Spacer;
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);

        XYDataset dataset = createDataset();

        JFreeChart chart = createChart(dataset);

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

    }

    private JFreeChart createChart(XYDataset dataset) {

        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            "Legal & General Unit Trust Prices",
            "Date", "Price Per Unit",
            dataset,
            true,
            true,
            false
        );

        chart.setBackgroundPaint(Color.white);

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

        XYPlot plot = chart.getXYPlot();
        plot.setBackgroundPaint(Color.lightGray);
```

```
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
        plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
        plot.setDomainCrosshairVisible(true);
        plot.setRangeCrosshairVisible(true);

        XYItemRenderer renderer = plot.getRenderer();
        if (renderer instanceof StandardXYItemRenderer) {
            StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
            rr.setPlotShapes(true);
            rr.setShapesFilled(true);
        }

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

        return chart;

    }

    private 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);

        dataset.setDomainIsPointsInTime(true);

        return dataset;

    }

    public static void main(String[] args) {

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

    }

}
```

Chapter 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 border;
- the chart title and sub-titles;
- the background color and/or image;
- the rendering hints that are used to draw the chart, including whether or not *anti-aliasing* is used;

These items are described in the following sections.

8.2.2 The Chart Border

JFreeChart can draw a border around the outside of a chart. By default, no border is drawn, but you can change this using the `setBorderVisible()` method. The color and line-style for the border are controlled by the `setBorderPaint()` and `setBorderStroke()` methods.

Note: if you are displaying your chart inside a `ChartPanel`, then you might prefer to use the border facilities provided by Swing.

8.2.3 The Chart Title

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

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

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

```
chart.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(RectangleEdge.BOTTOM);
```

If you prefer to have no title on your chart, you can set the title to `null`.

8.2.4 Subtitles

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

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

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:

```
Title subtitle = chart.getSubtitle(0);
```

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

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

8.2.5 Setting the Background Color

You can use the `setBackgroundPaint()` method to set the background color for a chart.[1] For example:

```
chart.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));
chart.setBackgroundPaint(p);
```

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

[1] 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.

8.2.6 Using a Background Image

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

```
chart.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.

```
chart.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).

Rendering Hints

JFreeChart uses the Java2D API to draw charts. Within this API, you can specify *rendering hints* to fine tune aspects of the way that the rendering engine works.

JFreeChart allows you to specify the rendering hints to be passed to the Java2D API when charts are drawn—use the `setRenderingHints()` method.

As a convenience, a method is provided to turn anti-aliasing on or off. With anti-aliasing on, charts appear to be smoother but they take longer to draw:

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

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

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 = chart.getPlot();
```

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

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

...or:

```
XYPlot plot = chart.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 = chart.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 = chart.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

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.

8.4.1 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 plot = chart.getCategoryPlot();
CategoryAxis domainAxis = plot.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 plot = chart.getXYPlot();
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setAutoRange(false);
```

8.4.2 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.

8.4.3 Rotating Axis Labels

When an axis is drawn at the left or right of a plot (a "vertical" axis), the label is automatically rotated by 90 degrees to minimise the space required. If you prefer to have the label drawn horizontally, you can change the label angle:

```
XYPlot plot = chart.getXYPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setLabelAngle(Math.PI / 2.0);
```

Note that the angle is specified in *radians* (`Math.PI` = 180 degrees).

8.4.4 Hiding Tick Labels

To hide the tick labels for an axis:

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

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

8.4.5 Hiding Tick Marks

To hide the tick marks for an axis:

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

Category axes do not have tick marks.

8.4.6 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.

8.4.7 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 plot = chart.getXYPlot();
NumberAxis axis = (NumberAxis) plot.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.

8.4.8 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.

Chapter 9

Dynamic Charts

9.1 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.

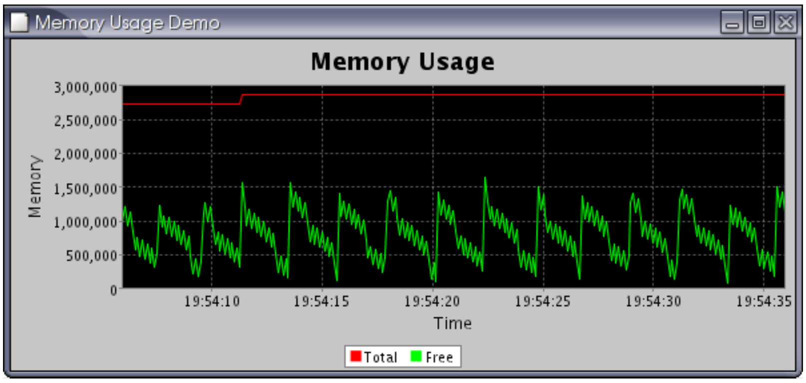


Figure 9.1: A dynamic chart demo

9.2 Background

9.2.1 Event notification

JFreeChart uses an *event notification mechanism* that allows it to respond to changes to any component of the chart.

For example, whenever a dataset is updated, a `DatasetChangeEvent` is sent to all listeners that are registered with the dataset. This triggers the following sequence of events:

- the plot (which registers itself with the dataset as a `DatasetChangeListener`) receives notification of the dataset change. It updates the axis ranges (if necessary) then passes on a PlotChangeEvent to all *its* registered listeners;

- the chart receives notification of the plot change event, and passes on a ChartChangeEvent to all *its* registered listeners;
- finally, for charts that are displayed in a ChartPanel, the panel will receive the chart change event. It responds by redrawing the chart—a complete redraw, not just the updated data.

A similar sequence of events happens for all changes to a chart or its subcomponents.

9.2.2 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*.

9.3 The Demo Application

9.3.1 Overview

The MemoryUsage.java demonstration is included in the "premium demos" download available to purchasers of this document. You can obtain this from:

http://www.object-refinery.com/jfreechart/premium/index.html

You will need to enter the username and password supplied with your original purchase of the JFreeChart Developer Guide.

9.3.2 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.

9.3.3 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`. It also helps to retain a reference to the dataset used in the chart.

9.3.4 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.

9.3.5 Source Code

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.DateAxis;
import org.jfree.chart.axis.NumberAxis;
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);

        DateAxis domain = new DateAxis("Time");
        NumberAxis range = new NumberAxis("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, 600, 280);
        frame.setVisible(true);
        panel.new DataGenerator().start();

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

}
```

Chapter 10

Tooltips

10.1 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;

10.2 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.

10.2.1 Pie Charts

The `PiePlot` class generates tool tips using the `PieToolTipGenerator` interface. A standard implementation (`StandardPieItemLabelGenerator`) 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.

10.2.2 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 `CategoryToolTipGenerator` interface specifies the method via which the renderer will obtain tool tips (if required).

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.

10.2.3 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 `XYItemLabelGenerator`.

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

10.3 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.

10.4 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.

10.5 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.

10.6 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.

Chapter 11

Item Labels

11.1 Introduction

11.1.1 Overview

For many chart types, JFreeChart will allow you to display *item labels* near each data item in a chart. For example, you can display the actual value represented by the bars in a bar chart—see figure 11.1.

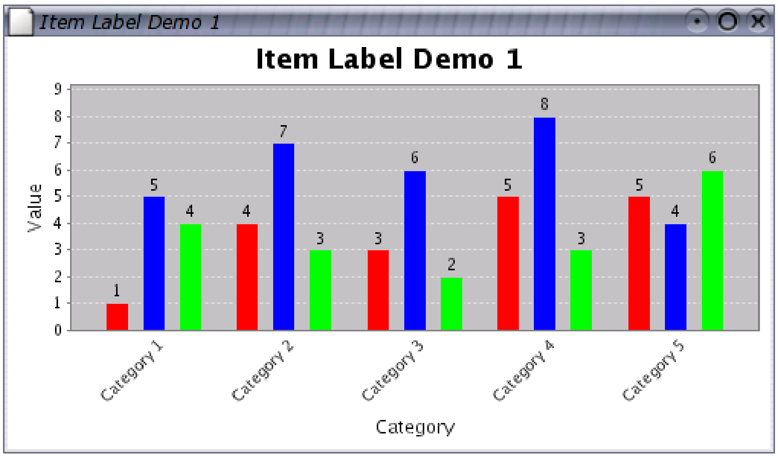


Figure 11.1: A bar chart with item labels

In this chapter, I describe how to:

- make item labels visible (for the chart types that support item labels);
- change the appearance (font and color) of item labels;
- specify the location of item labels;
- customise the item label text.

A word of advice—charts are supposed to *summarise* your data, so use this feature sparingly. If you feel it is necessary to display the actual data values all over your chart, then you might consider presenting your data in a tabular form instead.

11.1.2 Limitations

There are some limitations with respect to the item labels in the current release of JFreeChart:

- some renderers do not support item labels;
- axis ranges are not automatically adjusted to take into account the item labels—some labels may disappear off the chart if sufficient margins are not set (use the `setUpperMargin()` and/or `setLowerMargin()` methods in the relevant axis to adjust the settings).

In future releases, some or all of these limitations will be addressed.

11.2 Displaying Item Labels

11.2.1 Overview

Item labels are not visible by default, so you need to configure the renderer to create and display them. This involves two steps:

- assign a `CategoryItemLabelGenerator` to the renderer—this is an object that assumes responsibility for creating the labels;
- set a flag in the renderer to make the labels visible, either for *all series* or, if you prefer, on a *per series* basis.

These steps are detailed in the following sections. In addition, you have the option to customise the position of the item labels—this is described in section 11.4.

11.2.2 Assigning a Label Generator

Item labels are created by a `CategoryItemLabelGenerator` that is assigned to the renderer (the same mechanism is also used for tooltips). To assign a new generator, use the following code:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setLabelGenerator(new StandardCategoryItemLabelGenerator());
```

The `StandardCategoryItemLabelGenerator` is the only generator provided in the JFreeChart distribution. You can customise the behaviour of the standard generator via settings that you can apply in the constructor, or you can create your own generator as described in section 11.5.2.

11.2.3 Making Labels Visible For All Series

The following code will make item labels visible for the items in all series:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelsVisible(true);
```

Once set, this flag takes precedence over any *per series* settings you may have made elsewhere. In order for the per series settings to apply, you need to set this flag to `null` (see section 11.2.4).

11.2.4 Making Labels Visible For Selected Series

If you prefer, you can set flags that control the visibility of the item labels on a per series basis. For example, item labels are displayed only for the first series in figure 11.2.

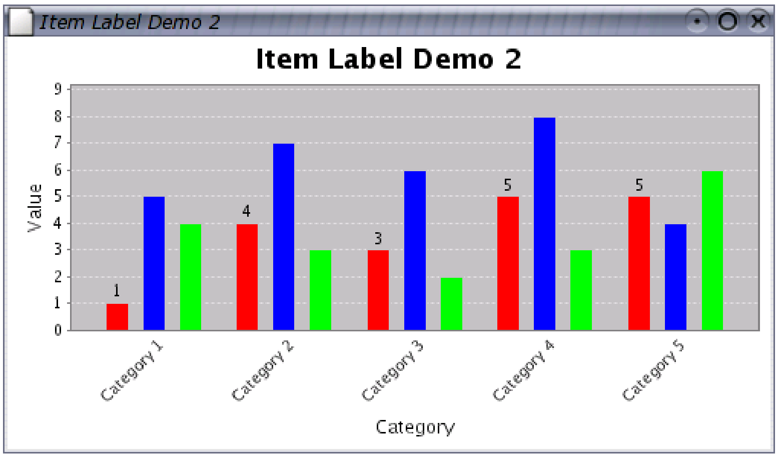


Figure 11.2: Item labels for selected series only

You can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelsVisible(null); // clears the ALL series flag
renderer.setSeriesItemLabelsVisible(0, true);
renderer.setSeriesItemLabelsVisible(1, false);
```

Notice that the flag for "all series" has been set to `null`—this is important, because the "all series" flag takes precedence over the "per series" flags.

11.2.5 Troubleshooting

If, after following the steps outlined in the previous sections, you still can't see any labels on your chart, there are a couple of things to consider:

- the renderer must have a `CategoryItemLabelGenerator` assigned to it—this is an object that creates the text items that are used for each label.
- some renderers don't yet support the display of item labels (refer to the documentation for the renderer you are using).

11.3 Item Label Appearance

11.3.1 Overview

You can change the appearance of the item labels by changing the font and/or the color used to display the labels. As for most other renderer attributes, the settings can be made once for *all series*, or on a *per series* basis.

In the current release of JFreeChart, labels are drawn with a transparent background. You cannot set a background color for the labels, nor can you specify that a border be drawn around the labels. This may change in the future.

11.3.2 Changing the Label Font

To change the font for the item labels in all series, you can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelFont(new Font("SansSerif", Font.PLAIN, 10));
```

Similarly, to set the font for individual series:

```
CategoryItemRenderer renderer = plot.getRenderer();

// clear the settings for ALL series...
renderer.setItemLabelFont(null);

// add settings for individual series...
renderer.setSeriesItemLabelFont(0, new Font("SansSerif", Font.PLAIN, 10));
renderer.setSeruesItemLabelFont(1, new Font("SansSerif", Font.BOLD, 10));
```

Notice how the font for all series has be set to `null` to prevent it from overriding the per series settings.

11.3.3 Changing the Label Color

To change the color for the item labels in all series, you can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelPaint(Color.red);
```

Similarly, to set the color for individual series:

```
CategoryItemRenderer renderer = plot.getRenderer();

// clear the settings for ALL series...
renderer.setItemLabelPaint(null);

// add settings for individual series...
renderer.setSeriesItemLabelPaint(0, Color.red);
renderer.setSeriesItemLabelPaint(1, Color.blue);
```

Once again, notice how the paint for all series has been set to `null` to prevent it from overriding the per series settings.

11.4 Item Label Positioning

11.4.1 Overview

The positioning of item labels is controlled by four attributes that are combined into an `ItemLabelPosition` object. You can define label positions for items with positive and negative values independently, via the following methods in the `CategoryItemRenderer` interface:

```
public void setPositiveItemLabelPosition(ItemLabelPosition position);
public void setNegativeItemLabelPosition(ItemLabelPosition position);
```

Understanding how these attributes impact the final position of individual labels is key to getting good results from the item label features in JFreeChart.

There are four attributes:

- the *item label anchor* - determines the base location for the item label;
- the *text anchor* - determines the point on the label that is aligned to the base location;
- the *rotation anchor* - this is the point on the label text about which the rotation (if any) is applied;
- the *rotation angle* - the angle through which the label is rotated.

These are described in the following sections.

11.4.2 The Item Label Anchor

The purpose of the item label anchor setting is to determine an *(x, y)* location on the chart that is near to the data item that is being labelled. The label is then aligned to this anchor point when it is being drawn. Refer to the `ItemLabelAnchor` documentation for more information.

11.4.3 The Text Anchor

The text anchor determines which point on the label should be aligned with the anchor point described in the previous section. It is possible to align the center of the label with the anchor point, or the top-right of the label, or the bottom-left, and so on...refer to the `TextAnchor` documentation for all the options.

Running the `DrawStringDemo` application in the `org.jfree.demo` package (included in the JCommon distribution) is a good way to gain an understanding of how the text anchor is used to align labels to a point on the screen.

11.4.4 The Rotation Anchor

The rotation anchor defines a point on the label about which the rotation (if any) will be applied to the label. The `DrawStringDemo` class also demonstrates this feature.

11.4.5 The Rotation Angle

The rotation angle defines the angle through which the label is rotated. The angle is specified in radians, and the rotation point is defined by the rotation anchor described in the previous section.

11.5 Customising the Item Label Text

11.5.1 Overview

Up to this point, we've relied on the label generator built in to JFreeChart to create the text for the item labels. If you want to have complete control over the label text, you can write your own class that implements the CategoryItemLabelGenerator interface.

In this section I provide a brief overview of the technique for implementing a custom label generator, then present two examples to illustrate the type of results you can achieve with this technique.

11.5.2 Implementing a Custom Label Generator

To develop a custom label generator, you simply need to write a class that implements the method defined in the CategoryItemLabelGenerator interface:

```
public String generateItemLabel(CategoryDataset dataset,
int series, int category);
```

The renderer will call each method at the point that it requires a `String` use for a label, and will pass in the CategoryDataset and the `series` and `category` indices for the current item. This means that you have full access to the entire dataset (not just the current item) for the creation of the label.

Both methods can return arbitrary `String` values, so you can apply any formatting you want to the results. It is also valid to return `null` if you prefer no label to be displayed.

All this is best illustrated by way of examples, which are provided in the following sections.

11.6 Example 1 - Values Above a Threshold

11.6.1 Overview

In this first example, the goal is to display labels for the items that have a value greater than some predefined threshold value (see figure 11.3).

It isn't all that difficult to achieve, we simply need to:

- write a class that implements the CategoryItemLabelGenerator interface, and implement the `generateItemLabel()` method in such a way that it returns `null` for any item where the value is less than the threshold;
- create an instance of this new class, and assign it to the renderer using the `setLabelGenerator()` method.

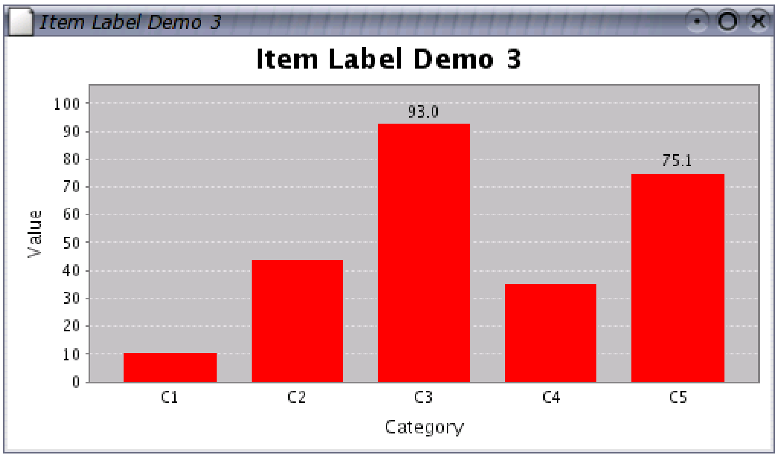


Figure 11.3: Item labels above a threshold

11.6.2 Source Code

The complete source code is presented below.

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

import java.awt.Color;
import java.awt.Dimension;

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.labels.CategoryItemLabelGenerator;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.CategoryItemRenderer;
import org.jfree.data.CategoryDataset;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demo showing a label generator that only displays labels for items
 * with a value that is greater than some threshold.
 */
public class ItemLabelDemo3 extends ApplicationFrame {

    /**
     * A custom label generator.
     */
    static class LabelGenerator implements CategoryItemLabelGenerator {

        /** The threshold. */
        private double threshold;

        /**
         * Creates a new generator that only displays labels that are greater
         * than or equal to the threshold value.
         *
         * @param threshold  the threshold value.
         */
        public LabelGenerator(double threshold) {
            this.threshold = threshold;
        }
```

```
    /**
     * Generates a label for the specified item. The label is typically a
     * formatted version of the data value, but any text can be used.
     *
     * @param dataset  the dataset (<code>null</code> not permitted).
     * @param series  the series index (zero-based).
     * @param category  the category index (zero-based).
     *
     * @return the label (possibly <code>null</code>).
     */
    public String generateItemLabel(CategoryDataset dataset,
                                    int series,
                                    int category) {

        String result = null;
        Number value = dataset.getValue(series, category);
        if (value != null) {
            double v = value.doubleValue();
            if (v > this.threshold) {
                result = value.toString();  // could apply formatting here
            }
        }
        return result;

    }

}

/**
 * Creates a new demo instance.
 *
 * @param title  the frame title.
 */
public ItemLabelDemo3(String title) {

    super(title);
    CategoryDataset dataset = createDataset();
    JFreeChart chart = createChart(dataset);
    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setPreferredSize(new Dimension(500, 270));
    setContentPane(chartPanel);

}

/**
 * Returns a sample dataset.
 *
 * @return the dataset.
 */
private CategoryDataset createDataset() {

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(11.0, "S1", "C1");
    dataset.addValue(44.3, "S1", "C2");
    dataset.addValue(93.0, "S1", "C3");
    dataset.addValue(35.6, "S1", "C4");
    dataset.addValue(75.1, "S1", "C5");
    return dataset;

}

/**
 * Creates a sample chart.
 *
 * @param dataset  the dataset.
 *
 * @return the chart.
 */
private JFreeChart createChart(CategoryDataset dataset) {

    // create the chart...
    JFreeChart chart = ChartFactory.createBarChart(
        "Item Label Demo 3",      // chart title
        "Category",               // domain axis label
```

```
            "Value",                  // range axis label
            dataset,                  // data
            PlotOrientation.VERTICAL, // orientation
            false,                    // include legend
            true,                     // tooltips?
            false                     // URLs?
        );

        chart.setBackgroundPaint(Color.white);

        CategoryPlot plot = chart.getCategoryPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);

        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setUpperMargin(0.15);

        CategoryItemRenderer renderer = plot.getRenderer();
        renderer.setItemLabelsVisible(true);
        renderer.setLabelGenerator(new LabelGenerator(50.0));

        return chart;

    }

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

        ItemLabelDemo3 demo = new ItemLabelDemo3("Item Label Demo 3");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

11.7 Example 2 - Displaying Percentages

11.7.1 Overview

In this example, the requirement is to display a bar chart where each bar is labelled with the value represented by the bar and also a percentage (where the percentage is calculated relative to a particular bar within the series OR the total of all the values in the series)—see figure 11.4.

In this implementation, the label generator calculates the percentage value on-the-fly. If a category index is supplied in the constructor, the base value used to calculate the percentage is taken from the specified category within the current series. If no category index is available, then the total of all the values in the current series is used as the base.

A default percentage formatter is created within the label generator—a more sophisticated implementation would provide the ability for the formatter to be customised via the generator's constructor.

11.7.2 Source Code

The complete source code follows.

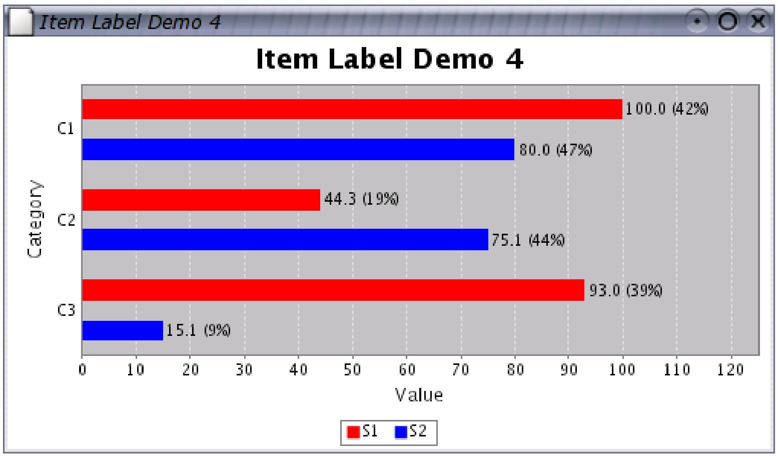


Figure 11.4: Percentage item labels

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

import java.awt.Color;
import java.awt.Dimension;
import java.text.NumberFormat;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.AxisLocation;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.labels.CategoryItemLabelGenerator;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.CategoryItemRenderer;
import org.jfree.data.CategoryDataset;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demo showing a label generator that displays labels that include
 * a percentage calculation.
 *
 */
public class ItemLabelDemo4 extends ApplicationFrame {

    /**
     * A custom label generator.
     */
    static class LabelGenerator implements CategoryItemLabelGenerator {

        /**
         * The index of the category on which to base the percentage
         * (null = use series total).
         */
        private Integer category;

        /** A percent formatter. */
        private NumberFormat formatter = NumberFormat.getPercentInstance();

        /**
         * Creates a new label generator that displays the item value and a
         * percentage relative to the value in the same series for the
         * specified category.
         *
```

```
     * @param category  the category index (zero-based).
     */
    public LabelGenerator(int category) {
        this(new Integer(category));
    }

    /**
     * Creates a new label generator that displays the item value and
     * a percentage relative to the value in the same series for the
     * specified category.  If the category index is <code>null</code>,
     * the total of all items in the series is used.
     *
     * @param category  the category index (<code>null</code> permitted).
     */
    public LabelGenerator(Integer category) {
        this.category = category;
    }

    /**
     * Generates a label for the specified item. The label is typically
     * a formatted version of the data value, but any text can be used.
     *
     * @param dataset  the dataset (<code>null</code> not permitted).
     * @param series  the series index (zero-based).
     * @param category  the category index (zero-based).
     *
     * @return the label (possibly <code>null</code>).
     */
    public String generateItemLabel(CategoryDataset dataset,
                                    int series,
                                    int category) {

        String result = null;
        double base = 0.0;
        if (this.category != null) {
            Number b = dataset.getValue(series, this.category.intValue());
            base = b.doubleValue();
        }
        else {
            base = calculateSeriesTotal(dataset, series);
        }
        Number value = dataset.getValue(series, category);
        if (value != null) {
            double v = value.doubleValue();
            // you could apply some formatting here
            result = value.toString()
                   + " (" + this.formatter.format(v / base) + ")";
        }
        return result;

    }

    private double calculateSeriesTotal(CategoryDataset dataset, int series) {
        double result = 0.0;
        for (int i = 0; i < dataset.getColumnCount(); i++) {
            Number value = dataset.getValue(series, i);
            if (value != null) {
                result = result + value.doubleValue();
            }
        }
        return result;
    }

}

/**
 * Creates a new demo instance.
 *
 * @param title  the frame title.
 */
public ItemLabelDemo4(String title) {

    super(title);
    CategoryDataset dataset = createDataset();
```

```
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new Dimension(500, 270));
        setContentPane(chartPanel);

    }

    /**
     * Returns a sample dataset.
     *
     * @return the dataset.
     */
    private CategoryDataset createDataset() {

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(100.0, "S1", "C1");
        dataset.addValue(44.3, "S1", "C2");
        dataset.addValue(93.0, "S1", "C3");
        dataset.addValue(80.0, "S2", "C1");
        dataset.addValue(75.1, "S2", "C2");
        dataset.addValue(15.1, "S2", "C3");
        return dataset;

    }

    /**
     * Creates a sample chart.
     *
     * @param dataset  the dataset.
     *
     * @return the chart.
     */
    private JFreeChart createChart(CategoryDataset dataset) {

        // create the chart...
        JFreeChart chart = ChartFactory.createBarChart(
            "Item Label Demo 4",        // chart title
            "Category",                 // domain axis label
            "Value",                    // range axis label
            dataset,                    // data
            PlotOrientation.HORIZONTAL, // orientation
            true,                       // include legend
            true,                       // tooltips?
            false                       // URLs?
        );

        chart.setBackgroundPaint(Color.white);

        CategoryPlot plot = chart.getCategoryPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
        plot.setRangeAxisLocation(AxisLocation.BOTTOM_OR_LEFT);

        NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setUpperMargin(0.25);

        CategoryItemRenderer renderer = plot.getRenderer();
        renderer.setItemLabelsVisible(true);

        // use one or the other of the following lines to see the different modes for
        // the label generator...
        renderer.setItemLabelGenerator(new LabelGenerator(null));
        //renderer.setItemLabelGenerator(new LabelGenerator(0));

        return chart;

    }

    /**
     * Starting point for the demonstration application.
     *
     * @param args  ignored.
     */
```

```
    public static void main(String[] args) {

        ItemLabelDemo4 demo = new ItemLabelDemo4("Item Label Demo 4");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);

    }

}
```

Chapter 12

Using Multiple Axes

12.1 Introduction

JFreeChart supports the use of *multiple axes* in the `CategoryPlot` and `XYPlot` classes. You can use this feature to display two or more datasets on a single chart, while making allowance for the fact that the datasets may contain data of vastly different magnitudes—see figure 12.1 for an example.

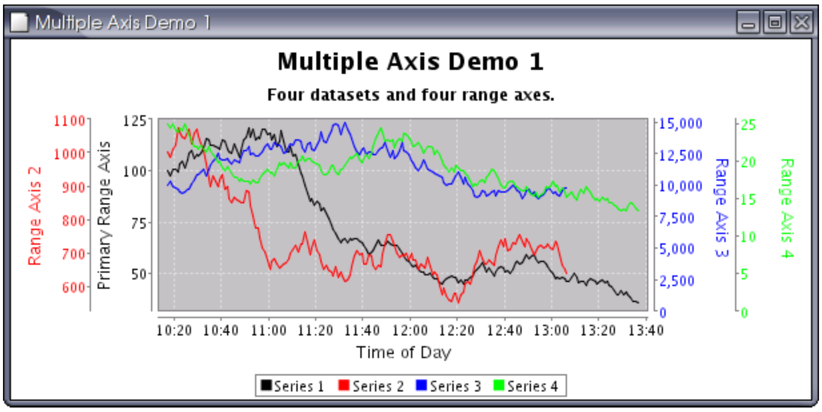


Figure 12.1: A chart with multiple axes

Typical charts constructed with JFreeChart use a *plot*, a *dataset*, a *renderer*, a *domain axis* and a *range axis*. To allow for multiple axis support, JFreeChart has been extended to allow for zero, one or many:

- *secondary datasets*;
- *secondary renderers*;
- *secondary domain axes*;
- *secondary range axes*;

In this section, several examples are presented showing how to use these additional datasets, renderers and axes.

12.2 An Example

12.2.1 Introduction

The `MultipleAxisDemo1.java` application (included in the JFreeChart distribution) provides a good example of how to create a chart with multiple axes. This section provides some notes on the steps taken within that code.

12.2.2 Create a Chart

To create a chart with multiple axes, datasets, and renderers, you should first create a regular chart (for example, using the `ChartFactory` class). You can use any chart that is constructed using a `CategoryPlot` or an `XYPlot`.

12.2.3 Add a Secondary Axis

You can create and add a secondary axis as follows:

```
NumberAxis axis2 = new NumberAxis("Range Axis 2");
plot.setSecondaryRangeAxis(0, axis2);
plot.setSecondaryRangeAxisLocation(0, AxisLocation.BOTTOM_OR_RIGHT);
```

In the code above, a `NumberAxis` with default settings has been used—you are free to customise the axis in the usual ways.

The `setSecondaryRangeAxis()` method is used to add the axis to the plot. Note that an index of `0` (zero) has been used—you can add as many secondary axes as you require, by incrementing the index each time you add a new axis.

The `setSecondaryRangeAxisLocation()` method allows you to specify where the axis will appear on the chart, using the `AxisLocation` class. You can have the axis on the same side as the primary axis, or on the opposite side—the choice is yours. In the example, `BOTTOM_OR_RIGHT` is specified, which means (for a range axis) on the right if the plot has a vertical orientation, or at the bottom if the plot has a horizontal orientation.

At this point, no secondary dataset has been added to the chart, so if you were to display the chart you would see the secondary axis, but it would have no data plotted against it.

12.2.4 Add a Secondary Dataset

Use the `setSecondaryDataset()` method to add a secondary dataset to the plot:

```
XYDataset dataset2 = ...  // up to you
plot.setSecondaryDataset(0, dataset2);
```

By default, the dataset will be plotted *against the primary range axis*. To have the dataset plotted against a secondary axis, use the `mapSecondaryDatasetToDomainAxis()` and `mapSecondaryDatasetToRangeAxis()` methods. These methods accept two arguments, the first is the index of the secondary dataset, and the second is an `Integer` where:

- a `null` value indicates the primary range axis;
- a non-`null` value indicates the index of the secondary range axis against which the dataset should be plotted.

12.2.5 Add a Secondary Renderer

Use the `setSecondaryRenderer()` method to add a renderer that will be used to plot the data from a secondary dataset:

```
XYItemRenderer renderer2 = ...  // up to you
plot.setSecondaryRenderer(0, renderer2);
```

The index (`0` in this case) should correspond to the index of the secondary axis and dataset added previously.

Note: if you don't specify a secondary renderer, the primary renderer will be used instead, but then the series colors will be shared between the primary dataset and the secondary dataset.

12.3 Hints and Tips

When using multiple axes, you need to provide some visual cue to readers to indicate which axis applies to a particular series. In the `MultipleAxisDemo1.java` application, the color of the axis label text has been changed to match the series color.

Additional demos included in the JFreeChart distribution include:

- `DualAxisDemo.java`
- `DualAxisDemo2.java`
- `DualAxisDemo3.java`
- `DualAxisDemo4.java`

Chapter 13

Combined Charts

13.1 Introduction

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

- CombinedDomainCategoryPlot / CombinedRangeCategoryPlot;
- CombinedDomainXYPlot / CombinedRangeXYPlot;

This section presents a few examples that use the combined chart facilities provided by JFreeChart. All the examples are included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

13.2 Combined Domain Category Plot

13.2.1 Overview

A *combined domain category plot* is a plot that displays two or more subplots (instances of CategoryPlot) that share a common *domain axis*. Each subplot maintains its own *range axis*. An example is shown in figure 13.1.

It is possible to display this chart with a horizontal or vertical orientation—the example shown has a vertical orientation.

13.2.2 Constructing the Chart

A demo application (`CombinedCategoryPlotDemo1.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) provides an example of how to create this type of chart. The key step is the creation of a CombinedDomainCategoryPlot instance, to which subplots are added:

```
CategoryAxis domainAxis = new CategoryAxis("Category");
CombinedDomainCategoryPlot plot = new CombinedDomainCategoryPlot(domainAxis);
plot.add(subplot1, 2);
plot.add(subplot2, 1);

JFreeChart result = new JFreeChart(
    "Combined Domain Category Plot Demo",
```

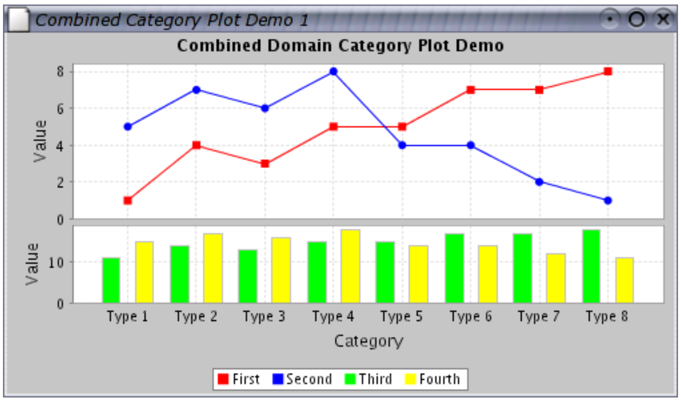


Figure 13.1: A combined domain category plot

```
        new Font("SansSerif", Font.BOLD, 12),
        plot,
        true
);
```

Notice how `subplot1` has been added with a weight of `2` (the second argument in the `add()` method, while `subplot2` has been added with a weight of `1`. This controls the amount of space allocated to each plot.

The subplots are regular `CategoryPlot` instances that have had their domain axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
CategoryDataset dataset1 = createDataset1();
NumberAxis rangeAxis1 = new NumberAxis("Value");
rangeAxis1.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
LineAndShapeRenderer renderer1 = new LineAndShapeRenderer();
renderer1.setBaseToolTipGenerator(new StandardCategoryItemLabelGenerator());
CategoryPlot subplot1 = new CategoryPlot(dataset1, null, rangeAxis1, renderer1);
subplot1.setDomainGridlinesVisible(true);

CategoryDataset dataset2 = createDataset2();
NumberAxis rangeAxis2 = new NumberAxis("Value");
rangeAxis2.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
BarRenderer renderer2 = new BarRenderer();
renderer2.setBaseToolTipGenerator(new StandardCategoryItemLabelGenerator());
CategoryPlot subplot2 = new CategoryPlot(dataset2, null, rangeAxis2, renderer2);
subplot2.setDomainGridlinesVisible(true);
```

13.3 Combined Range Category Plot

13.3.1 Overview

A *combined range category plot* is a plot that displays two or more subplots (instances of `CategoryPlot`) that share a common *range axis*. Each subplot maintains its own *domain axis*. An example is shown in figure 13.2.

It is possible to display this chart with a horizontal or vertical orientation (the example above has a vertical orientation).

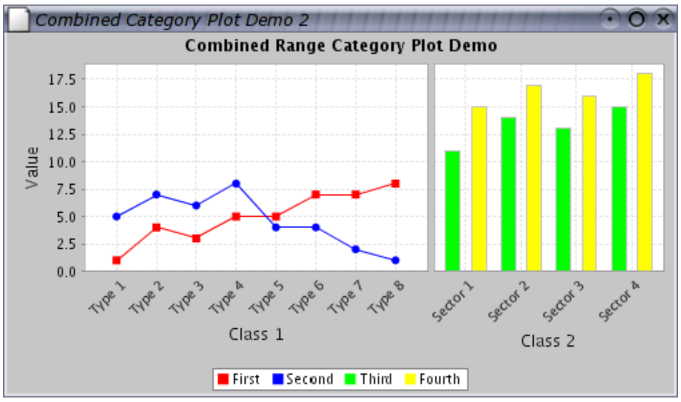


Figure 13.2: A combined range category plot.

13.3.2 Constructing the Chart

A demo application (`CombinedCategoryPlotDemo2.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) provides an example of how to create this type of chart. The key step is the creation of a `CombinedRangeCategoryPlot` instance, to which subplots are added:

```
ValueAxis rangeAxis = new NumberAxis("Value");
CombinedRangeCategoryPlot plot = new CombinedRangeCategoryPlot(rangeAxis);
plot.add(subplot1, 3);
plot.add(subplot2, 2);

JFreeChart result = new JFreeChart(
    "Combined Range Category Plot Demo",
    new Font("SansSerif", Font.BOLD, 12),
    plot,
    true
);
```

Notice how `subplot1` has been added with a weight of `3` (the second argument in the `add()` method), while `subplot2` has been added with a weight of `2`. This controls the amount of space allocated to each plot.

The subplots are regular `CategoryPlot` instances that have had their range axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
CategoryDataset dataset1 = createDataset1();
CategoryAxis domainAxis1 = new CategoryAxis("Class 1");
domainAxis1.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
domainAxis1.setMaxCategoryLabelWidthRatio(5.0f);
LineAndShapeRenderer renderer1 = new LineAndShapeRenderer();
renderer1.setBaseToolTipGenerator(new StandardCategoryItemLabelGenerator());
CategoryPlot subplot1 = new CategoryPlot(dataset1, domainAxis1, null, renderer1);
subplot1.setDomainGridlinesVisible(true);

CategoryDataset dataset2 = createDataset2();
CategoryAxis domainAxis2 = new CategoryAxis("Class 2");
domainAxis2.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
domainAxis2.setMaxCategoryLabelWidthRatio(5.0f);
BarRenderer renderer2 = new BarRenderer();
renderer2.setBaseToolTipGenerator(new StandardCategoryItemLabelGenerator());
CategoryPlot subplot2 = new CategoryPlot(dataset2, domainAxis2, null, renderer2);
subplot2.setDomainGridlinesVisible(true);
```

13.4 Combined Domain XY Plot

13.4.1 Overview

A *combined domain XY plot* is a plot that displays two or more subplots (instances of XYPlot) that share a common *domain axis*. Each subplot maintains its own *range axis*. An example is shown in figure 13.3.

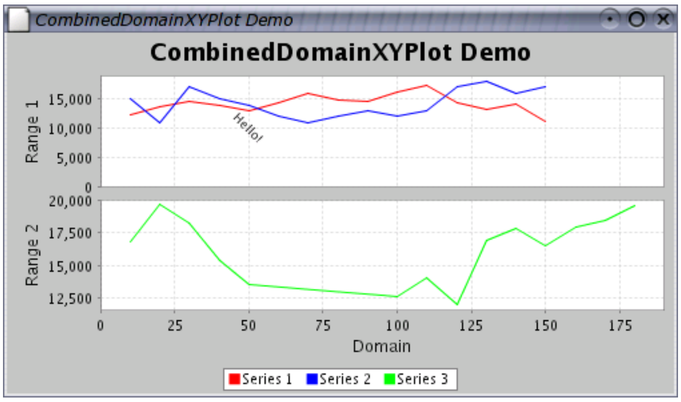


Figure 13.3: A combined domain XY plot

It is possible to display this chart with a horizontal or vertical orientation (the example shown has a vertical orientation).

13.4.2 Constructing the Chart

A demo application (`CombinedXYPlotDemo1.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) provides an example of how to create this type of chart. The key step is the creation of a CombinedDomainXYPlot instance, to which subplots are added:

```
CombinedDomainXYPlot plot = new CombinedDomainXYPlot(new NumberAxis("Domain"));
plot.setGap(10.0);

plot.add(subplot1, 1);
plot.add(subplot2, 1);
plot.setOrientation(PlotOrientation.VERTICAL);

return new JFreeChart(
    "CombinedDomainXYPlot Demo",
    JFreeChart.DEFAULT_TITLE_FONT, plot, true
);
```

Notice how the subplots are added with weights (both 1 in this case). This controls the amount of space allocated to each plot.

The subplots are regular XYPlot instances that have had their domain axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
XYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new StandardXYItemRenderer();
NumberAxis rangeAxis1 = new NumberAxis("Range 1");
```

```
XYPlot subplot1 = new XYPlot(data1, null, rangeAxis1, renderer1);
subplot1.setRangeAxisLocation(AxisLocation.BOTTOM_OR_LEFT);

XYTextAnnotation annotation = new XYTextAnnotation("Hello!", 50.0, 10000.0);
annotation.setFont(new Font("SansSerif", Font.PLAIN, 9));
annotation.setRotationAngle(Math.PI / 4.0);
subplot1.addAnnotation(annotation);

// create subplot 2...
XYDataset data2 = createDataset2();
XYItemRenderer renderer2 = new StandardXYItemRenderer();
NumberAxis rangeAxis2 = new NumberAxis("Range 2");
rangeAxis2.setAutoRangeIncludesZero(false);
XYPlot subplot2 = new XYPlot(data2, null, rangeAxis2, renderer2);
subplot2.setRangeAxisLocation(AxisLocation.TOP_OR_LEFT);
```

13.5 Combined Range XY Plot

13.5.1 Overview

A *combined range XY plot* is a plot that displays two or more subplots (instances of `XYPlot`) that share a common *range axis*. Each subplot maintains its own *domain axis*. An example is shown in figure 13.4.

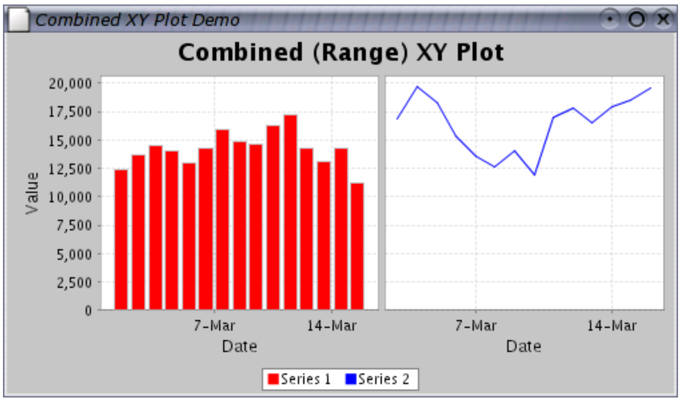


Figure 13.4: A combined range XY plot

It is possible to display this chart with a horizontal or vertical orientation (the example shown has a vertical orientation).

13.5.2 Constructing the Chart

A demo application (`CombinedXYPlotDemo2.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) provides an example of how to create this type of chart. The key step is the creation of a `CombinedRangeXYPlot` instance, to which subplots are added:

```
// create the plot...
CombinedRangeXYPlot plot = new CombinedRangeXYPlot(new NumberAxis("Value"));
plot.add(subplot1, 1);
plot.add(subplot2, 1);

return new JFreeChart(
```

```
    "Combined (Range) XY Plot",
    JFreeChart.DEFAULT_TITLE_FONT, plot, true
);
```

Notice how the subplots are added with weights (both 1 in this case). This controls the amount of space allocated to each plot.

The subplots are regular `XYPlot` instances that have had their range axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
// create subplot 1...
IntervalXYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new XYBarRenderer(0.20);
renderer1.setToolTipGenerator(
    new StandardXYItemLabelGenerator(
        new SimpleDateFormat("d-MMM-yyyy"), new DecimalFormat("0,000.0")
    )
);
XYPlot subplot1 = new XYPlot(data1, new DateAxis("Date"), null, renderer1);

// create subplot 2...
XYDataset data2 = createDataset2();
XYItemRenderer renderer2 = new StandardXYItemRenderer();
renderer2.setToolTipGenerator(
    new StandardXYItemLabelGenerator(
        new SimpleDateFormat("d-MMM-yyyy"), new DecimalFormat("0,000.0")
    )
);
XYPlot subplot2 = new XYPlot(data2, new DateAxis("Date"), null, renderer2);
```

Chapter 14

Datasets and JDBC

14.1 Introduction

In this section, I describe the use of several *datasets* that are designed to work with JDBC to obtain data from database tables:

- `JDBCPieDataset`
- `JDBCCategoryDataset`
- `JDBCXYDataset`

These datasets have been developed by Bryan Scott of the Australian Antarctic Division.

14.2 About JDBC

JDBC is a high-level Java API for working with relational databases. JDBC does a good job of furthering Java's *platform independence*, making it possible to write portable code that will work with many different database systems.

JDBC provides a mechanism for loading a *JDBC driver* specific to the database system actually being used. JDBC drivers are available for many databases, on many different platforms.

14.3 Sample Data

To see the JDBC datasets in action, you need to create some sample data in a test database.

Here is listed some sample data that will be used to create a pie chart, a bar chart and a time series chart.

A pie chart will be created using this data (in a table called `piedata1`):

```
CATEGORY | VALUE
---------+------
London   |  54.3
```

```
New York |  43.4
Paris    |  17.9
```

Similarly, a bar chart will be created using this data (in a table called `categorydata1`):

```
CATEGORY | SERIES1 | SERIES2 | SERIES3
---------+---------+---------+--------
London   |    54.3 |    32.1 |    53.4
New York |    43.4 |    54.3 |    75.2
Paris    |    17.9 |    34.8 |    37.1
```

Finally, a time series chart will be generated using this data (in a table called `xydata1`):

```
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.

14.4 PostgreSQL

14.4.1 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.

14.4.2 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.

14.4.3 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);
```

14.4.4 Creating the Category Chart Data

To create the table for the category dataset:

```
CREATE TABLE categorydata1 (
    category VARCHAR(32),
    series1  FLOAT,
    series2  FLOAT,
    series3  FLOAT
);
```

...and to populate it:

```
INSERT INTO categorydata1 VALUES ('London',   54.3, 32.1, 53.4);
INSERT INTO categorydata1 VALUES ('New York', 43.4, 54.3, 75.2);
INSERT INTO categorydata1 VALUES ('Paris',    17.9, 34.8, 37.1);
```

14.4.5 Creating the XY Chart Data

To create the table for the XY dataset:

```
CREATE TABLE xydata1 (
    date     DATE,
    series1  FLOAT,
    series2  FLOAT,
    series3  FLOAT
);
```

...and to populate it:

```
INSERT INTO xydata1 VALUES ('1-Aug-2002', 54.3, 32.1, 53.4);
INSERT INTO xydata1 VALUES ('2-Aug-2002', 43.4, 54.3, 75.2);
INSERT INTO xydata1 VALUES ('3-Aug-2002', 39.6, 55.9, 37.1);
INSERT INTO xydata1 VALUES ('4-Aug-2002', 35.4, 55.2, 27.5);
INSERT INTO xydata1 VALUES ('5-Aug-2002', 33.9, 49.8, 22.3);
INSERT INTO xydata1 VALUES ('6-Aug-2002', 35.2, 48.4, 17.7);
INSERT INTO xydata1 VALUES ('7-Aug-2002', 38.9, 49.7, 15.3);
INSERT INTO xydata1 VALUES ('8-Aug-2002', 36.3, 44.4, 12.1);
INSERT INTO xydata1 VALUES ('9-Aug-2002', 31.0, 46.3, 11.0);
```

Granting Table Permissions

The last step in setting up the sample database is to grant read access to the new tables to the user `jfreechart`:

```
GRANT SELECT ON piedata1 TO jfreechart;
GRANT SELECT ON categorydata1 TO jfreechart;
GRANT SELECT ON xydata1 TO jfreechart;
```

14.5 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.

14.6 The Demo Applications

14.6.1 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).

14.6.2 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).

14.6.3 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).

Chapter 15

Exporting Charts to Acrobat PDF

15.1 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.

15.2 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.

15.3 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 `1.01`.

15.4 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.

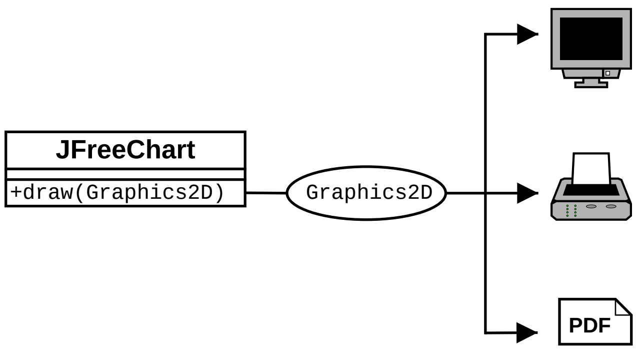


Figure 15.1: The JFreeChart `draw(...)` method

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.

15.5 Getting Started

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

File:	Description:
`jfreechart-0.9.18.jar`	The JFreeChart class library.
`jcommon-0.9.3.jar`	The JCommon class library (used by JFreeChart).
`iText-1.01.jar`	The iText class library.

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

15.6 The Application

The first thing the sample application needs to do is create a chart. Here we create a time series chart:

```
// create a chart...
XYDataset dataset = createDataset();
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Legal & General Unit Trust Prices",
    "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);
        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);
            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. * *
     *
```

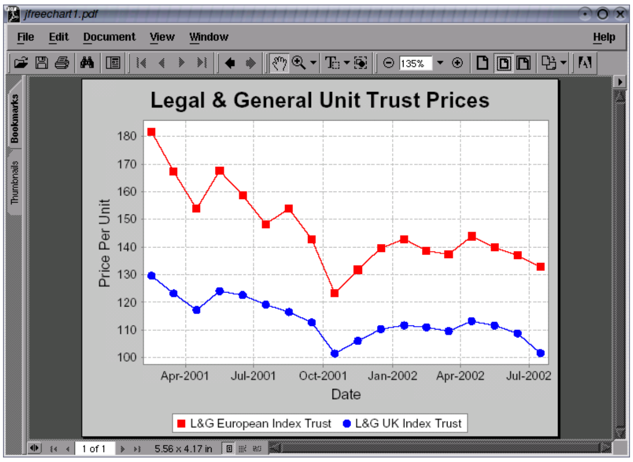
```
 * @return the dataset.
 */
public static XYDataset createDataset() {

    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;
}

public static void main(String[] args) {
    try {
        // create a chart...
        XYDataset dataset = createDataset();
        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            "Legal & General Unit Trust Prices",
            "Date",
            "Price Per Unit",
            dataset,
            true,
            true,
            false
         );

        // some additional chart customisation here...
        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.setShapesFilled(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 15.5 on your classpath.

15.7 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.

15.8 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.

15.8.1 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.slovo.info/unifonts.htm`

I have tried out the `tahoma.ttf` font with success. In fact, I will use this font in the example that follows. The Tahoma font doesn't support every character defined in Unicode, so if you have specific requirements then you need to choose an appropriate font. At one point I had the Arial Unicode MS font (`arialuni.ttf`)installed on my system—this has support for the full Unicode character set, although this means that the font definition file is quite large (around 24 megabytes!)

15.8.2 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.

15.8.3 Mapping Additional Fonts

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

`/usr/lib/SunJava2/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/SunJava2/jre/lib/fonts");
DefaultFontMapper.BaseFontParameters pp =
                      mapper.getBaseFontParameters("Tahoma");
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";
//String text = "hi";
Font font = new Font("Tahoma", Font.PLAIN, 12);
TextTitle subtitle = new TextTitle(text, font);
 chart.addSubtitle(subtitle);
```

Notice that the subtitle (a random 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 15.2. 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.

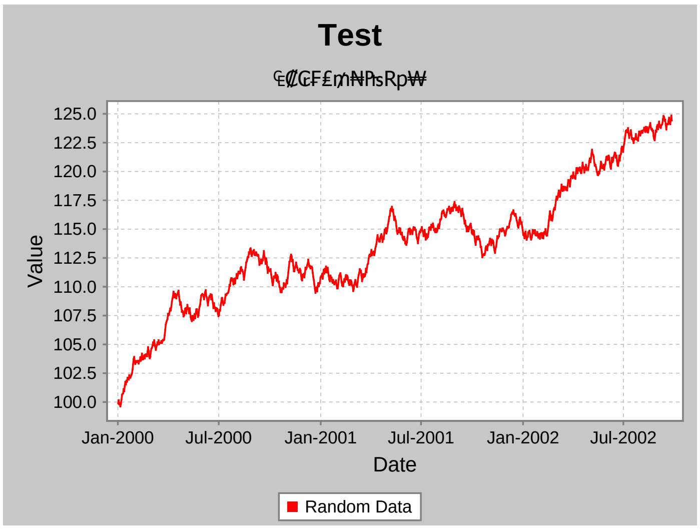


Figure 15.2: A Unicode subtitle

Chapter 16

Exporting Charts to SVG Format

16.1 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).

16.2 Background

16.2.1 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).

16.2.2 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 `1.5`.

16.3 A Sample Application

16.3.1 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.

16.3.2 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-0.9.3.jar`	Common classes from The Object Refinery.
`jfreechart-0.9.18.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.

16.3.3 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.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.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.DefaultPieDataset;
import org.w3c.dom.DOMImplementation;
import org.w3c.dom.Document;

/**
 * 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);

        // set the precision to avoid a null pointer exception in Batik 1.5
        svgGenerator.getGeneratorContext().setPrecision(6);

        // 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);

    }

}
```

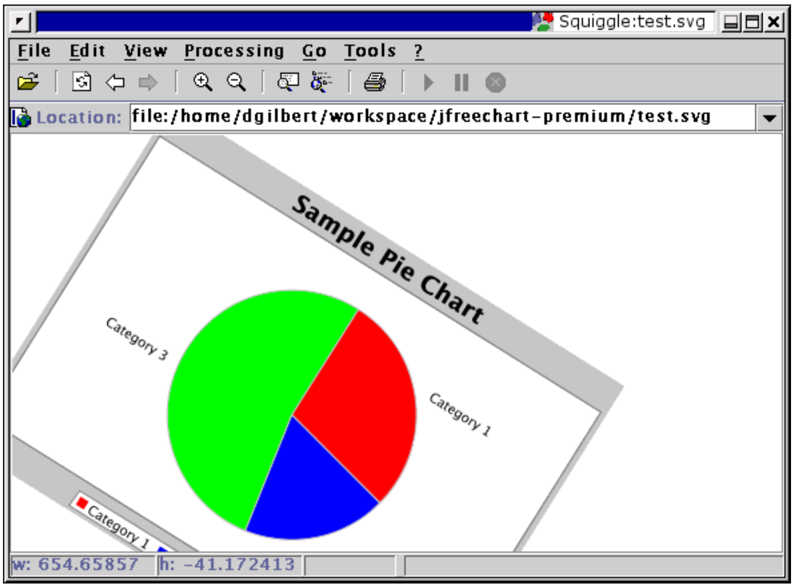
Running this program creates a file `test.svg` in SVG format.

16.3.4 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-squiggle.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.

Chapter 17

Applets

17.1 Introduction

Subject to a couple of provisos, using JFreeChart in an applet is relatively straightforward. This section provides a brief overview of the important issues and describes a working example that should be sufficient to get you started.

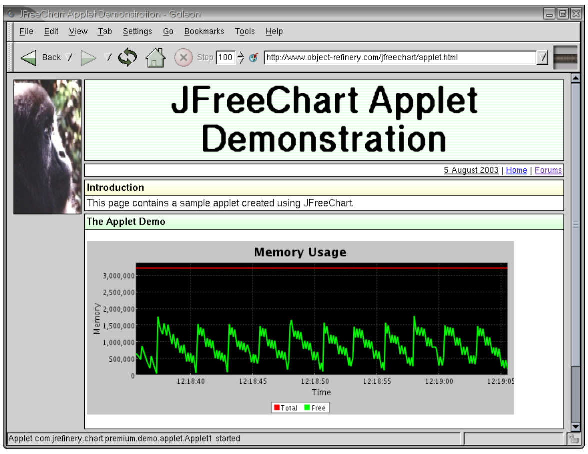


Figure 17.1: An applet using JFreeChart

Figure 17.1 shows a sample applet that uses JFreeChart. This applet is available online at:

```
http://www.object-refinery.com/jfreechart/applet.html
```

The source code for this applet appears later in this section.

17.2 Issues

The main issues to consider when developing applets (whether with or without JFreeChart) are:

- browser support;
- security restrictions;
- code size.

Be sure that you understand these issues *before* you commit significant resources to writing applets.

17.2.1 Browser Support

The *vast majority of web browsers* provide support for the latest version of Java (JDK 1.4) and will therefore have no problems running applets that use JFreeChart (recall that JFreeChart will run on any version of the JDK from 1.2.2 onwards).

However, the *vast majority of users* on the web use (by default in most cases) the one web browser—Microsoft Internet Explorer (MSIE)—that only supports a version of Java (JDK 1.1) that is now hopelessly out-of-date. This is a problem, because applets that use JFreeChart will not work on a default installation of MSIE. There is a workaround—users can download and install Sun's Java plugin—but, like many workarounds, it is too much effort and inconvenience for many people. The end result is a deployment problem for developers who choose to write applets.

This single issue has caused many developers to abandon their plans to develop applets[1] and instead choose an easier-to-deploy technology such as *Java Servlets* (see the next chapter).

17.2.2 Security

Applets (and Java more generally) have been designed with security in mind. When an applet runs in your web browser, it is restricted in the operations that it is permitted to perform. For example, an applet typically will not be allowed to read or write to the local filesystem. Describing the details of Java's security mechanism is beyond the scope of this text, but you should be aware that some functions provided by JFreeChart (for example, the option to save charts to PNG format via the pop-up menu) will not work in applets that are subject to the default security policy. If you need these functions to work, then you will need to study Java's security mechanism in more detail.

17.2.3 Code Size

A final issue to consider is the size of the "runtime" code required for your applet. Before an applet can run, the code (typically packed into jar files) has to be downloaded to the end user's computer. Clearly, for users with limited bandwidth connections, the size of the code can be an issue.

[1] For some people this issue won't be a concern. For example, you may be developing applets for internal corporate use, and your standard desktop configuration includes a browser that supports JDK 1.4. Alternatively, you may be providing an applet for public use via the World Wide Web, but it is not critical that *every* user be able to run the applet.

The JFreeChart code is distributed in a jar file that is around 500KB in size. That isn't large—especially when you consider the number and variety of charts that JFreeChart supports—but, at the same time, it isn't exactly optimal for a user on a dial-up modem connection. And you need to add to that the JCommon jar file (around 170KB) plus whatever code you have for your applet.

As always with JFreeChart, you have the source code so you could improve this by repackaging the JFreeChart jar file to include only those classes that are used by your applet (directly or indirectly).

17.3 A Sample Applet

As mentioned in the introduction, a sample applet that uses JFreeChart can be seen at the following URL:[2]

```
http://www.object-refinery.com/jfreechart/applet.html
```

Two aspects of the sample applet are interesting, the source code that is used to create the applet and the HTML file that is used to invoke the applet.

17.3.1 The HTML

The HTML used to invoke the applet is important, since it needs to reference the necessary jar files. The HTML applet tag used is:

```
<APPLET ARCHIVE="jfreechart-0.9.4-premium-demo-applets.jar,jfreechart-0.9.4.jar,
jcommon-0.7.1.jar" CODE="com.jrefinery.chart.premium.demo.applet.Applet1"
width=640 height=260 ALT="You should see an applet, not this text.">
</APPLET>
```

Notice that three jar files are referenced. The first contains the applet class (source code in the next section) only, while the remaining two jar files are the standard JFreeChart and JCommon class libraries (the version numbers reflect the age of the demo rather than the current releases).

You can place the applet tag anywhere in your HTML file that you might place some other element (such as an image).

17.3.2 The Source Code

The sample applet is created using the following source code (which is included in the "support demos" package). There is very little applet-specific code here—we just extend `JApplet`:

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

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JApplet;
import javax.swing.Timer;

import org.jfree.chart.ChartPanel;
```

[2] If the applet does not work for you, please check that your web browser is configured correctly and supports JDK 1.2.2 or later.

```
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.DefaultXYItemRenderer;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;

public class Applet1 extends JApplet {

    /** Time series for total memory used. */
    private TimeSeries total;

    /** Time series for free memory. */
    private TimeSeries free;

    public Applet1() {

        // 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);

        DateAxis domain = new DateAxis("Time");
        NumberAxis range = new NumberAxis("Memory");

        XYPlot xyplot = new XYPlot(dataset, domain, range, new DefaultXYItemRenderer());
        xyplot.setBackgroundPaint(Color.black);
        xyplot.getRenderer().setSeriesPaint(0, Color.red);
        xyplot.getRenderer().setSeriesPaint(1, Color.blue);

        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);
        chartPanel.setPopupMenu(null);

        getContentPane().add(chartPanel);
        new Applet1.DataGenerator().start();

    }

    /**
     * Adds an observation to the 'total memory' time series.
     *
     * @param y  the total memory used.
     */
    private void addTotalObservation(double y) {
        total.add(new Millisecond(), y);
    }

    /**
     * Adds an observation to the 'free memory' time series.
     *
     * @param y  the free memory.
     */
    private void addFreeObservation(double y) {
        free.add(new Millisecond(), y);
    }

    /**
     * The data generator.
     */
```

```
    class DataGenerator extends Timer implements ActionListener {

        /**
         * Constructor.
         */
        DataGenerator() {
            super(100, null);
            addActionListener(this);
        }

        /**
         * Adds a new free/total memory reading to the dataset.
         *
         * @param event  the action event.
         */
        public void actionPerformed(ActionEvent event) {
            long f = Runtime.getRuntime().freeMemory();
            long t = Runtime.getRuntime().totalMemory();
            addTotalObservation(t);
            addFreeObservation(f);
        }

    }

}
```

Chapter 18

Servlets

18.1 Introduction

The *Java Servlets API* is a very popular technology for creating web applications. JFreeChart is well suited for use in a servlet environment and, in this section, some examples are presented to help those developers that are interested in using JFreeChart for web applications.

All the sample code in this section is available for download from the same page as the JFreeChart Developer Guide:

`http://www.object-refinery.com/jfreechart/premium/index.html`

The file to download is `jfreechart-0.9.18-premium-demos.zip`.[1]

18.2 A Simple Servlet

The `ServletDemo1` class implements a very simple servlet that returns a PNG image of a bar chart generated using JFreeChart. When it is run, the servlet will return a raw image to the client (web browser) which will display the 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 the example is:

- a good illustration of the *request-response* nature of servlets;
- useful as a test case if you are configuring a server environment and want to check that everything is working.

We will move on to a more complex example later, showing how to request different charts using HTML forms, and embedding the generated charts within HTML output.

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

[1] To access this page you need to enter the username and password provided to you in the confirmation e-mail you received when you purchased the JFreeChart Priority Support package.

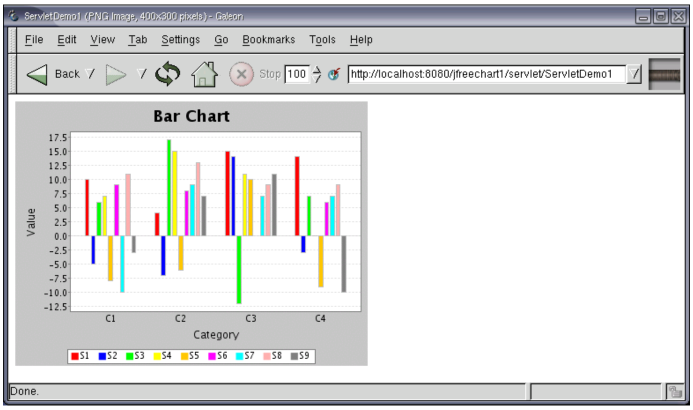


Figure 18.1: `ServletDemo1` in a browser

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

import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.DefaultCategoryDataset;

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.createBarChart(
                    "Bar Chart",
                    "Category",
                    "Value",
                    dataset,
                    PlotOrientation.VERTICAL,
                    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). You can find out more about Tomcat at:

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

You will also require the JFreeChart and JCommon jar files to compile the above servlet.

18.3 Deploying the Servlet

Servlets are deployed in the `webapps` directory provided by your servlet engine. In my case, I am using Tomcat 4.1.18 on SUSE Linux 8.2, and the directory is:[2]

```
/opt/jakarta/tomcat/webapps
```

Within the `webapps` directory, create a `jfreechart1` directory to hold the first servlet demo, then create the following structure within the directory:

```
.../jfreechart1/WEB-INF/web.xml
.../jfreechart1/WEB-INF/lib/jfreechart-0.9.13.jar
.../jfreechart1/WEB-INF/lib/jcommon-0.8.8.jar
.../jfreechart1/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo1.class
```

You need to create the `web.xml` file—it provides information about the servlet:

```
<?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.demo.ServletDemo1
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/servlet/ServletDemo1</url-pattern>
  </servlet-mapping>
</web-app>
```

Once you have all these files in place, restart your servlet engine and type in the following URL using your favourite web browser:

```
http://localhost:8080/jfreechart1/servlet/ServletDemo1
```

If all is well, you will see the chart image displayed in your browser, as shown in figure 18.1.

18.4 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 available in the `jfreechart-0.9.13-premium-demos.zip` file.

`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, as shown in figure 18.2.

Here is the code for `ServletDemo2`:

[2] Servlets are portable between different servlet engines, so if you are using a different servlet engine, consult the documentation to find the location of the `webapps` folder.

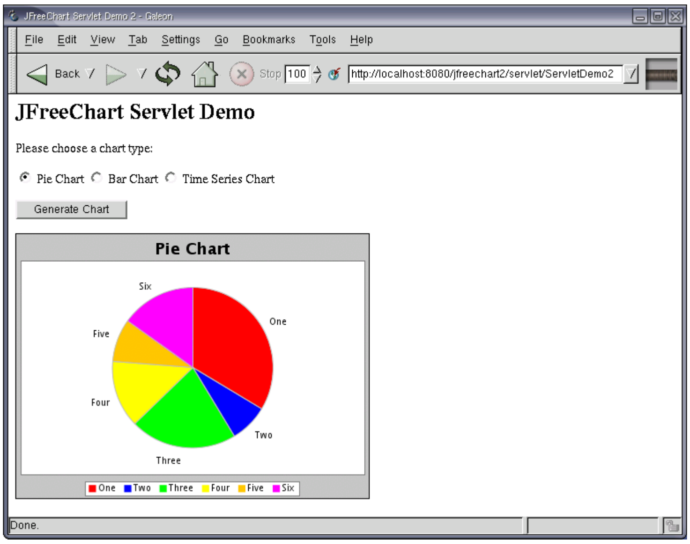


Figure 18.2: `ServletDemo2` in a browser

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

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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\""
```

```
                    + timeCheckled + "> 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).[3]

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.

Here is the source code for `ServletDemo2ChartGenerator`:

```
public class ServletDemo2ChartGenerator extends HttpServlet {

    public ServletDemo2ChartGenerator() {
    }

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

        OutputStream out = response.getOutputStream();
        try {
            String type = request.getParameter("type");
            JFreeChart chart = null;
            if (type.equals("pie")) {
                chart = createPieChart();
            }
            else if (type.equals("bar")) {
                chart = createBarChart();
            }
            else if (type.equals("time")) {
                chart = createTimeSeriesChart();
            }
            if (chart != null) {
                response.setContentType("image/png");
                ChartUtilities.writeChartAsPNG(out, chart, 400, 300);
            }
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
```

[3]The `Writer` is wrapped in a `PrintWriter` in order to use the more convenient methods available in the latter class.

```
        finally {
            out.close();
        }

    }

    private JFreeChart createPieChart() {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();
        data.setValue("One", new Double(43.2));
        data.setValue("Two", new Double(10.0));
        data.setValue("Three", new Double(27.5));
        data.setValue("Four", new Double(17.5));
        data.setValue("Five", new Double(11.0));
        data.setValue("Six", new Double(19.4));

        JFreeChart chart = ChartFactory.createPieChart(
            "Pie Chart",
            data,
            true, true, false
        );
        return chart;

    }

    private JFreeChart createBarChart() {

        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.createBarChart3D(
            "Bar Chart",
            "Category",
            "Value",
            dataset,
            PlotOrientation.VERTICAL,
            true,
            true,
```

```
            false
        );
        return chart;

    }

    private JFreeChart createTimeSeriesChart() {

        // here we just populate a series with random data...
        TimeSeries series = new TimeSeries("Random Data");
        Day current = new Day(1, SerialDate.JANUARY, 2001);
        for (int i = 0; i < 100; i++) {
            series.add(current, Math.random() * 100);
            current = (Day) current.next();
        }
        XYDataset data = new TimeSeriesCollection(series);

        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            "Time Series Chart",
            "Date",
            "Rate",
            data,
            true,
            true,
            false
        );
        return chart;

    }
}
```

18.5 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 `index.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, `chart.html`:

```
<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.

18.6 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.demo.ServletDemo1
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2
    </servlet-name>
    <servlet-class>
      com.jrefinery.chart.demo.ServletDemo2
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2ChartGenerator
    </servlet-name>
    <servlet-class>
      com.jrefinery.chart.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 `jfreechart2` directory within Tomcat's `webapps` directory. The `index.html` and `chart.html` files are copied to this directory.

```
webapps/jfreechart2/index.html
webapps/jfreechart2/chart.html
```

Next, a subdirectory WEB-INF is created within the `jfreechart2` directory, and the `web.xml` file is copied to here.

```
webapps/jfreechart2/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/jfreechart2/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo1.class
webapps/jfreechart2/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo2.class
webapps/jfreechart2/WEB-INF/classes/com/jrefinery/chart/demo/ServletDemo2ChartGenerator.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/jfreechart2/WEB-INF/lib/jcommon-0.8.8.jar
webapps/jfreechart2/WEB-INF/lib/jfreechart-0.9.13.jar
```

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

```
http://localhost:8080/jfreechart2/index.html
```

If all the files have been put in the correct places, you should see the running servlet demonstration (this has been tested using Tomcat 4.1.18 running on SuSE Linux 8.2).

Chapter 19

Miscellaneous

19.1 Introduction

This section contains miscellaneous information about JFreeChart.

19.2 X11 / Headless Java

If you are using JFreeChart in a server environment running Unix / Linux, you may encounter the problem that JFreeChart won't run without X11. This is a common problem for Java code that relies on AWT, see the following web page for further information:

`http://java.sun.com/products/java-media/2D/forDevelopers/java2dfaq.html#xvfb`

There is also a thread in the JFreeChart forum with lots of info:

`http://www.jfree.org/phpBB2/viewtopic.php?t=1012`

19.3 Java Server Pages

Developers that are interested in using JFreeChart with JSP will want to check out the Cewolf project:

`http://cewolf.sourceforge.net/`

Thanks to Guido Laures for leading this effort.

Chapter 20

Packages

20.1 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.imagemap	HTML image map utility classes.
o.j.chart.labels	The item label and tooltip 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.title	Chart title 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.gantt	Dataset interfaces and classes for Gantt charts.
o.j.data.statistics	Classes that are used for generating statistics.
o.j.data.time	Time-based dataset interfaces and classes.

Additional information can be found in the Javadoc HTML files.

Chapter 21

Package: org.jfree.chart

21.1 Overview

This package contains the major classes and interfaces in the *JFreeChart Class Library*, including the all important `JFreeChart` class.

21.2 ChartColor

21.2.1 Overview

This class defines some standard colors.

21.2.2 Notes

The `DefaultDrawingSupplier` uses the `createDefaultPaintArray()` method to generate the default paint sequence for charts.

21.3 ChartFactory

21.3.1 Overview

This class contains a range of convenient methods for creating standard types of charts.

HINT: The use of these methods is optional. Take a look at the source code for the method you are using to see if it might be a better option to cut-and-paste the code into your application, and then customise it to meet your requirements.

21.3.2 Pie Charts

To create a regular pie chart:

```
public static JFreeChart createPieChart(String title,
PieDataset dataset, boolean legend, boolean tooltips, boolean urls);
```

Creates a *pie chart* for the specified `PieDataset` (`null` permitted). The chart is constructed using a `PiePlot`.

To create a pie chart with a "3D effect":

`public static JFreeChart createPieChart3D(String title, PieDataset dataset, boolean legend, boolean tooltips, boolean urls)`
Creates a *3D pie chart* for the specified `PieDataset` (`null` permitted). The chart is constructed using a `PiePlot3D`.

To create a single chart containing multiple pie charts:

`public static JFreeChart createMultiplePieChart(String title, CategoryDataset dataset, TableOrder order, boolean legend, boolean tooltips, boolean urls);`
Creates a *multiple pie chart* for the specified `CategoryDataset`. This chart is constructed using a `MultiplePiePlot`. The `order` argument can be either `TableOrder.BY_ROW` or `TableOrder.BY_COLUMN`.

To create a single chart containing multiple pie charts with a "3D effect":

`public static JFreeChart createMultiplePieChart3D(String title, CategoryDataset dataset, TableOrder order, boolean legend, boolean tooltips, boolean urls);`
Creates a *multiple pie chart* for the specified `CategoryDataset`. This chart is constructed using a `MultiplePiePlot`. The `order` argument can be either `TableOrder.BY_ROW` or `TableOrder.BY_COLUMN`.

21.3.3 Methods

To create a bar chart:

`public static JFreeChart createBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
Creates a horizontal or vertical *bar chart* for the given `CategoryDataset` (see the `BarRenderer` class documentation for an example).

To create a bar chart with a "3D effect":

`public static JFreeChart createBarChart3D(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
Creates a *bar chart with 3D effect* for the given `CategoryDataset` (see the `BarRenderer3D` class documentation for an example).

To create a stacked bar chart:

`public static JFreeChart createStackedBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
Creates a *stacked bar chart* for the given `CategoryDataset`.

To create a stacked bar chart with a "3D effect":

> `public static JFreeChart createStackedBarChart3D(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
> Creates a *stacked bar chart with 3D effect* for the given `CategoryDataset`.

To create a line chart based on a `CategoryDataset`:

> `public static JFreeChart createLineChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
> Creates a *line chart* for the given `CategoryDataset`.

To create a line chart based on a `XYDataset`:

> `public static JFreeChart createXYLineChart(String title, String xAxisLabel, String yAxisLabel, XYDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)`
> Creates a *XY line chart* for the given `XYDataset`.

To create a scatter plot:

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

To create a time series chart:

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

To create a bar chart using an `IntervalXYDataset` (bearing in mind that you can use the `XYBarDataset` wrapper to convert any `XYDataset` to the required type):

> `public static JFreeChart createXYBarChart(String title, String xAxisLabel, boolean dateAxis, String yAxisLabel, IntervalXYDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);`
> Creates an *XY bar chart* for the given `IntervalXYDataset`. The `dateAxis` argument allows you to select whether the chart is created with a `DateAxis` or a `NumberAxis` for the domain axis. The chart created with this method uses a `XYPlot` and `XYBarRenderer`.

To create a high-low-open-close chart:

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

To create a candlestick chart:

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

To create an area chart using data from a `XYDataset`:

> `public static JFreeChart createXYAreaChart(String title, String xAxisLabel, String yAxisLabel, XYDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)`
> Creates an area chart for the specified dataset. The chart that is created uses a XYPlot and a XYAreaRenderer.

To create a stacked area chart using data from a `TableXYDataset`:

> `public static JFreeChart createStackedXYAreaChart(String title, String xAxisLabel, String yAxisLabel, TableXYDataset dataset, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls)`
> Creates a stacked area chart for the specified dataset (notice that a `TableXYDataset` is required to enable stacking). The chart that is created uses a XYPlot and a StackedXYAreaRenderer.

21.4 ChartFrame

21.4.1 Overview

A frame containing chart within a `ChartPanel`.

21.4.2 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.

21.5 ChartMouseEvent

21.5.1 Overview

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

21.5.2 Notes

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

21.6 ChartMouseListener

21.6.1 Overview

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

21.6.2 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.

21.6.3 Notes

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

21.7 ChartPanel

21.7.1 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.

21.7.2 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 panel for displaying 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).

21.7.3 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.

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.

21.7.4 Tooltips

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.

The panel uses the standard Swing tool tip mechanism, which means that the tool tip timings (initial delay, dismiss delay and reshow delay) can be controlled application-wide using the usual Swing API calls. In addition, the panel has a facility to temporarily override the application wide settings while the mouse pointer is within the bounds of the panel:

> `public void setInitialDelay(int delay);`
> Sets the initial delay (in milliseconds) before tool tips are displayed.

> `public void setDismissDelay(int delay);`
> Sets the delay (in milliseconds) before tool tips are dismissed.

> `public void setReshowDelay(int delay);`
> Sets the delay (in milliseconds) before tool tips are reshown.

21.7.5 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`.

21.8 ChartPanelConstants

21.8.1 Overview

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

21.9 ChartRenderingInfo

21.9.1 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.

21.9.2 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.

21.9.3 Notes

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

21.10 ChartUtilities

21.10.1 Overview

This class contains utility methods for:

- creating images from charts—supported formats are PNG and JPEG;
- generating HTML image maps.

All of the methods in this class are `static`

21.10.2 Generating PNG Images

The *Portable Network Graphics* (PNG) format is a good choice for creating chart images. The format offers:

- a free and open specification;
- fast and effective compression;

- no loss of quality when images are reconstructed from the compressed binary format;
- excellent support in most web clients;

JFreeChart provides support for writing charts in PNG format via an encoder developed by J. David Eisenberg (published as free software under the terms of the GNU LGPL). You can find this encoder at:

```
http://www.catcode.com
```

The most general method allows you to write the image data directly to an output stream:

```
public static void writeChartAsPNG(OutputStream out, JFreeChart chart,
int width, int height) throws IOException
```
Writes a chart image of the specified size directly to the output stream.

If you need to retain information about the chart dimensions and content (to create an HTML image map, for example) you can pass in a newly created `ChartRenderingInfo` object using this method:

```
public static void writeChartAsPNG(OutputStream out, JFreeChart chart,
int width, int height, ChartRenderingInfo info)
```
Writes a chart image of the specified size directly to the output stream, and collects chart information in the supplied `info` object.

The above methods have counterparts that write image data directly to a file:

```
public static void saveChartAsPNG(File file, JFreeChart chart, int width,
int height);
```
Saves a chart image of the specified size into the specified file, using the PNG format.

```
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.

21.10.3 Generating JPEG Images

The *Joint Photographic Experts Group* (JPEG) image format is supported using methods that are almost identical to those listed for PNG in the previous section.

> *NOTE: JPEG is not an ideal format for charts. Images lose some definition after decompression from this format. This is most noticeable in high color contrast areas, which are common in charts. It is recommended that you use PNG format instead of JPEG, if at all possible.*

To write a chart to a file in 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.

21.10.4 HTML Image Maps

An *HTML image map* is an HTML fragment used to describe the characteristics of an image file. The image map can define regions within the image, and associate these with URLs and tooltip information.

To generate a simple HTML image map for a `JFreeChart` instance, first generate an image for the chart and be sure to retain the `ChartRenderingInfo` object from the image drawing. Then, generate the image map using the following method:

> `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`.

21.10.5 Notes

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

21.11 ClipPath

21.11.1 Overview

Not yet documented.

21.12 DrawableLegendItem

21.12.1 Overview

Used to represent a `LegendItem` plus it's physical drawing characteristics (position, label location etc.) as it is being laid out on the chart.

21.13 Effect3D

21.13.1 Overview

An interface that should be implemented by renderers that use a "3D effect". This allows the 3D axis classes to synchronise their own "3D effect" with that of the renderer and plot.

See Also

BarRenderer3D, CategoryAxis3D, NumberAxis3D.

21.14 JFreeChart

21.14.1 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, the `JFreeChart` class 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 (not all plot types use renderers).

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 `XYBarRenderer`.
`HighLowDataset`	`XYPlot` with a `HighLowRenderer`.
`HighLowDataset`	`XYPlot` with a `CandlestickRenderer`.

21.14.2 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.

21.14.3 Attributes

The attributes maintained by the `JFreeChart` class are listed in Table 21.1.

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 21.1: Attributes for the `JFreeChart` class

21.14.4 Methods

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

`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).

To set the title for a chart:

`public void setTitle(String title);`
Sets the title for a chart and sends a `ChartChangeEvent` to all registered listeners.

An alternative method for setting the chart title is:

`public void setTitle(TextTitle title);`
Sets the title for a chart and sends a `ChartChangeEvent` to all registered listeners.

Although a chart can have only one title, it can have any number of subtitles:

`public void addSubtitle(Title 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 set the background paint for the chart:

`public void setBackgroundPaint(Paint paint);`
Sets the background paint for the chart and sends a `ChartChangeEvent` to all registered listeners. If this is set to `null`, the chart background will be transparent.

You can set an optional background image for the chart:[1]

`public void setBackgroundImage(Image image);`
Sets the background image for the chart (`null` permitted) and sends a `ChartChangeEvent` to all registered listeners.

You need to ensure that the image supplied to the above method is fully loaded, see this link for more details:

`http://java.sun.com/docs/books/tutorial/uiswing/painting/loadingImages.html`

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.

[1] As an alternative to this method, note that you can set a background image for the chart's `Plot`. This will be positioned within the plot area only rather than the entire chart area.

21.14.5 Creating Images

The `JFreeChart` class includes utility methods for creating a `BufferedImage` containing the chart:

> `public BufferedImage createBufferedImage(int width, int height);`
> Creates a buffered image containing the chart. The size of the image is specified by the `width` and `height` arguments.

> `public BufferedImage createBufferedImage(int width, int height, ChartRenderingInfo info);`
> Creates a buffered image containing the chart. The size of the image is specified by the `width` and `height` arguments. The `info` argument is used to collect information about the chart as it is being drawn (required if you want to create an HTML image map for the image).

One other variation draws the chart at one size then scales it (up or down) to fit a different image size:

> `public BufferedImage createBufferedImage(int imageWidth, int imageHeight, double drawWidth, double drawHeight, ChartRenderingInfo info)`
> Creates an image containing a chart that has been drawn at one size then scaled (up or down) to fit the image size.

21.14.6 Notes

Some points to note:

- 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).

21.15 JFreeChartConstants

21.15.1 Overview

A collection of constants used by the `JFreeChart` class.

21.16 Legend

21.16.1 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.

21.16.2 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);
```

21.16.3 Constructor

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

21.16.4 Notes

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

See Also

`StandardLegend`.

21.17 LegendItem

21.17.1 Overview

An item within a legend.

21.18 LegendItemCollection

21.18.1 Overview

A collection of legend items.

See Also

`Legend`.

21.19 LegendItemLayout

21.19.1 Overview

An interface for laying out a collection of legend items.

Notes

This code is incomplete.

See Also

Legend.

21.20 MeterLegend

21.20.1 Overview

To be documented.

21.21 PolarChartPanel

21.21.1 Overview

An extension of the `ChartPanel` class with a pop-up menu that applies to polar charts.

21.22 StandardLegend

21.22.1 Overview

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

21.22.2 Methods

The legend position is controlled using methods inherited from the `Legend` class.

To set the color and thickness of the legend outline, use the following methods:

> `public void setOutlineStroke(Stroke stroke);`
> Sets the `Stroke` used to draw the outline for the legend and sends a `LegendChangeEvent` to all registered listeners.

> `public void setOutlinePaint(Paint paint);`
> Sets the `Paint` used to draw the outline for the legend and sends a `LegendChangeEvent` to all registered listeners.

To set the background color for the legend:

> `public void setBackgroundPaint(Paint paint);`
> Sets the `Paint` used to fill the background of the legend and sends a `LegendChangeEvent` to all registered listeners.

To set the title (optional) and title font for the legend:

> `public void setTitle(String title);`
> Sets the title for the legend.

> `public void setTitleFont(Font font);`
> Sets the title font for the legend and sends a `LegendChangeEvent` to all registered listeners.

To set the color and font used for the legend item text:

`public void setItemFont(Font font);`
Sets the font used to display the text for the legend items.

`public void setItemPaint(Paint paint);`
Sets the paint used to display the text for the legend items.

21.22.3 Legend Item Shapes

If your chart displays shapes to represent the items in a series, you can get the legend to reflect this using the following method:

`public void setDisplaySeriesShapes(boolean flag);`
Sets the flag that controls whether shapes are displayed for the legend items.

A range of methods are available to change the appearance of the shapes in the legend. The fill color is obtained from the chart's renderer, but the outline paint and stroke is set in the legend:

`public void setShapeOutlinePaint(Paint paint);`
Sets the `Paint` used to outline shapes in the legend.

`public void setShapeOutlineStroke(Stroke stroke);`
Sets the `Stroke` used to outline shapes in the legend.

You can also scale the size of the shapes displayed in the legend:

`public void setShapeScaleX(double factor);`
Sets the x scale factor for the shapes displayed in the legend.

`public void setShapeScaleY(double factor);`
Sets the y scale factor for the shapes displayed in the legend.

21.22.4 Notes

Some points to note:

- the legend does not have methods to get or set the items that will be displayed. At the time a chart is drawn, the legend items are obtained via a call to the `getLegendItems()` method in the `Plot` class;
- 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.

21.23 StandardLegendItemLayout

21.23.1 Overview

This class is not currently used.

Chapter 22

Package: org.jfree.chart.annotations

22.1 Overview

The annotations framework provides a mechanism for adding small text and graphics items to charts, usually to highlight a particular data item. In the current release, annotations can be added to the `CategoryPlot` and `XYPlot` classes. This framework is relatively basic at present, additional features are likely to be added in the future.

22.2 CategoryAnnotation

22.2.1 Overview

The interface that must be supported by annotations that are to be added to a `CategoryPlot`.

The `CategoryTextAnnotation` class is the only implementation of this interface that is included in the JFreeChart distribution.

22.2.2 Methods

This interface defines a single method:

> `public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis);`
> Draws the annotation.

22.3 CategoryTextAnnotation

22.3.1 Overview

An annotation that can be used to display an item of text at some location (defined by a *(category, value)* pair) on a `CategoryPlot`.

22.4 TextAnnotation

22.4.1 Overview

The base class for a *text annotation*. The class includes font, paint, alignment and rotation settings. Subclasses will add location information to the content represented by this class.

22.4.2 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);`
> Creates a new text annotation with the specified attributes.

22.4.3 Methods

There are methods for accessing the `text`, `font`, `paint`, `anchor` and `rotation` attributes.

22.4.4 Notes

`CategoryTextAnnotation` and `XYTextAnnotation` are the two subclasses included in the JFreeChart distribution.

22.5 XYAnnotation

22.5.1 Overview

The interface that must be supported by annotations that are to be added to an `XYPlot`.

This interface is implemented by:

- `XYDrawableAnnotation`;
- `XYLineAnnotation`;
- `XYPointerAnnotation`;
- `XYTextAnnotation`;

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

22.5.2 Methods

This class defines one method for drawing the annotation:

> `public void draw(Graphics2D g2, Rectangle2D dataArea,`
> `XYPlot plot, 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.

22.6 XYDrawableAnnotation

22.6.1 Overview

An annotation that draws an object at some *(x, y)* location on an XYPlot. The object can be any implementation of the Drawable interface (defined in the JCommon class library).

22.6.2 Notes

See the MarkerDemo1.java source file in the JFreeChart distribution for an example.

22.7 XYLineAnnotation

22.7.1 Overview

A simple annotation that draws a line between a starting point *(x0, y0)* and an ending point *(x1, y1)* on an XYPlot.

22.8 XYPointerAnnotation

22.8.1 Overview

An annotation that displays an arrow pointing towards a specific *(x, y)* location on an XYPlot. The arrow can have a label at one end. For example:

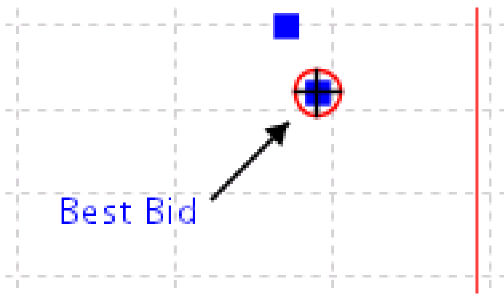


Figure 22.1: An XYPointerAnnotation example

22.8.2 Usage

To add a pointer annotation to an XYPlot:

```
XYPlot plot = myChart.getXYPlot();
XYPointerAnnotation pointer = new XYPointerAnnotation(
    "Best Bid", millis, 163.0, 3.0 * Math.PI / 4.0
);
pointer.setTipRadius(10.0);
pointer.setBaseRadius(35.0);
pointer.setFont(new Font("SansSerif", Font.PLAIN, 9));
```

```
pointer.setPaint(Color.blue);
pointer.setTextAnchor(TextAnchor.HALF_ASCENT_RIGHT);
plot.addAnnotation(pointer);
```

22.9 XYTextAnnotation

22.9.1 Overview

A text annotation that can be added to an `XYPlot`. You can use this class to add a small text label at some *(x, y)* location on a chart.

The annotation inherits font, paint, alignment and rotation settings from the `TextAnnotation` class.

22.9.2 Usage

To add a simple annotation to an `XYPlot`:

```
XYPlot plot = myChart.getXYPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

22.9.3 Methods

This class defines methods to get and set the `x` and `y` values (defining the location of the annotation against the domain and range axes).

22.9.4 Notes

Some points to note:

- the `XYPointerAnnotation` subclass can be used to display a label with an arrow pointing to some *(x, y)* value.

Chapter 23

Package: org.jfree.chart.axis

23.1 Overview

This package contains all the axis classes plus a few assorted support classes and interfaces:

- the `CategoryPlot` and `XYPlot` classes maintain references to two axes (by default), which we refer to as the *domain axis* and *range axis*. These terms are based on the idea that these plots are providing a visual representation of a function that maps a set of *domain values* onto a set of *range values*. For most purposes, you can think of the domain axis as the *X-axis* and the range axis as the *Y-axis*, but we prefer the more generic terms.

- the default settings provided by the axis classes should work well for a wide range of applications. However, there are many ways to customise the behaviour of the axes by modifying attributes via the JFreeChart API. Be sure to read through the API documentation to become familiar with the options that are available.

- a powerful feature of JFreeChart is the support for multiple domain and range axes on a single plot. If you plan to make use of this feature, you should refer to section 12 for more information.

The axis classes are `Cloneable` and `Serializable`.

23.2 Axis

23.2.1 Overview

An abstract base class representing an axis. Some subclasses of `Plot`, including `CategoryPlot` and `XYPlot`, will use axes to display data.

Figure 23.1 illustrates the axis class hierarchy.

23.2.2 Constructors

The constructors for this class are `protected`, you cannot create an instance of this class directly—you must use a subclass.

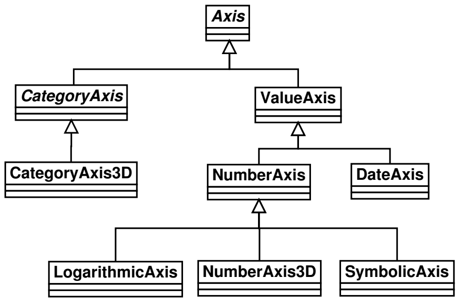


Figure 23.1: Axis classes

23.2.3 Attributes

The attributes maintained by the `Axis` class are listed in Table 23.1. 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.

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.
axisLineVisible	A flag that controls whether or not a line is drawn for the axis.
axisLinePaint	The paint used to draw the axis line if it is visible.
axisLineStroke	The stroke used to draw the axis line if it is visible.
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 23.1: Attributes for the `Axis` class

The default values used to initialise the axis attributes are listed in Table 23.2.

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 23.2: `Axis` class default attribute values

23.2.4 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:

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

23.2.5 Methods

All axes are drawn by the plot that owns the axis, using this method:

```
public abstract AxisState draw(Graphics2D g2, double cursor,
Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);
```

Draws the axis along the specified edge of the data area. Given that there may be more than one axis on a particular edge, the cursor value specifies the distance from the edge that the axis should be drawn (to take account of other axes that have already been drawn). An `AxisState` object is returned which provides information about the axis (for example, the tick values which the plot will use to draw gridlines if they are visible).

All axes are given the opportunity to refresh the axis ticks during the drawing process, which allows for dynamic adjustment depending on the amount of space available for drawing the axis:

```
public abstract List refreshTicks(Graphics2D g2, AxisState state,
Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);
```

Creates a list of ticks for the axis and updates the axis state.

23.2.6 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`.

23.3 AxisCollection

23.3.1 Overview

A storage structure that is used to record the axes that have been assigned to the top, bottom, left and right sides of a plot.

23.3.2 Notes

Axis collections are maintained only temporarily during the process of drawing a chart.

23.4 AxisConstants

23.4.1 Overview

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

23.4.2 Notes

The `Plot` class also implements this interface, so that it has convenient access to the constants for internal use.

23.5 AxisLocation

23.5.1 Overview

This class is used to represent the possible axis locations for a 2D chart:

- `AxisLocation.TOP_OR_LEFT`;
- `AxisLocation.TOP_OR_RIGHT`;
- `AxisLocation.BOTTOM_OR_LEFT`;
- `AxisLocation.BOTTOM_OR_RIGHT`;

The final position of the axis is dependent on the orientation of the plot (horizontal or vertical) and whether the axis is being used as a domain or a range axis.

23.5.2 Notes

The axis location is set using methods in the `CategoryPlot` and `XYPlot` classes.

23.6 AxisSpace

23.6.1 Overview

This class is used to record the amount of space (in Java2D units) required to display the axes around the edges of a plot. Since the plot may contain many axes (or, in the most complex case, many subplots containing many axes) this class is used to collate the space requirements for all the axes.

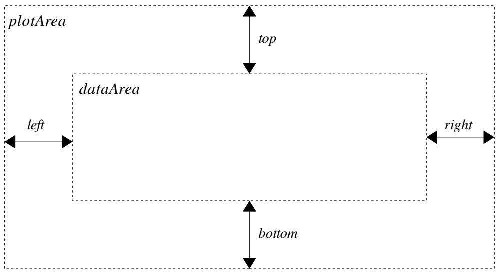


Figure 23.2: AxisSpace Attributes

Axes are always drawn around the edges of the *data area* but should never extend outside the *plot area*.

23.6.2 Methods

There are methods to get and set each of the attributes `top`, `bottom`, `left` and `right` maintained by this class.

To *add* space to a particular edge:

```
public void add(double space, RectangleEdge edge);
```
Adds the specified amount of space (in Java2D units) to one edge.

Sometimes you want to ensure that there is *at least* a specified amount of space for the axis along a particular edge (this is used to ensure that the data areas in combined plots are aligned). The following methods achieve this:

```
public void ensureAtLeast(double space, RectangleEdge edge);
```
Ensures that there is at least the specified amount of space for the axes along the specified edge.

`public void ensureAtLeast(AxisSpace space);`
As above, but applied to all the edges.

Given a rectangle and an instance of `AxisSpace`, you can calculate the size of an inner rectangle (essentially this is how the data area is computed from the plot area):

`public Rectangle2D shrink(Rectangle2D area, Rectangle2D result);`
Calculates an inner rectangle based on the current space settings. If `result` is `null` a new `Rectangle2D` is created for the result, otherwise the supplied rectangle is recycled.

23.7 AxisState

23.7.1 Overview

Instances of this class are used to record state information for an axis during the process of drawing the axis to some output target.

23.7.2 Notes

By recording state information *per drawing* of an axis, it should be possible for separate threads to draw the same axis to different output targets simultaneously without interfering with one another. This is part of an effort to (eventually) make JFreeChart thread-safe.

23.8 CategoryAnchor

23.8.1 Overview

An enumeration of the anchor points within the space allocated for a single category on a `CategoryAxis`:

Default:	**Value:**
`CategoryAnchor.START`	The start of the category.
`CategoryAnchor.MIDDLE`	The middle of the category.
`CategoryAnchor.END`	The end of the category.

23.8.2 Usage

This class is used to control the position of the domain axis gridlines drawn in a `CategoryPlot` (see the `setDomainGridlinePosition()` method).

23.9 CategoryAxis

23.9.1 Overview

A *category axis* is used as the domain axis in a `CategoryPlot`. Categories are displayed at regular intervals along the axis, with a gap before the first category (the *lower margin*), a gap after the last category (the *upper margin*) and a gap between each category (the *category margin*).

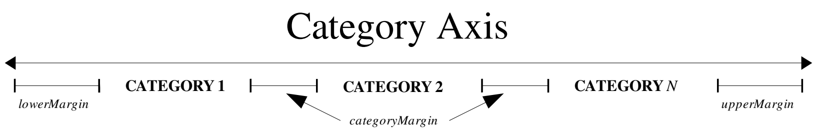


Figure 23.3: The CategoryAxis margins

The axis will usually display a label for each category. There are a range of options for controlling the position, alignment and rotation of the labels—these are described in section 23.9.5.

23.9.2 Constructor

There is a single constructor:

> `public CategoryAxis(String label);`
> Creates a new category axis with the specified label. If you prefer no axis label, you can use `null` for the `label` argument.

23.9.3 Attributes

The attributes maintained by the `CategoryAxis` class are listed in Table 23.3. These attributes are in addition to those inherited from the `Axis` class (see section 23.2.3 for details).

Attribute:	Description:
lowerMargin	The margin that appears before the first category, expressed as a percentage of the overall axis length (defaults to `0.05` or five percent).
upperMargin	The margin that appears after the last category, expressed as a percentage of the overall axis length (defaults to `0.05` or five percent).
categoryMargin	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). The default value is `0.20` (twenty percent).
categoryLabelPositionOffset	The offset between the axis line and the category labels.
categoryLabelPositions	A structure that defines label positioning information for each possible axis location (the axis may be located at the top, bottom, left or right of the plot).

Table 23.3: 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).

23.9.4 Setting Axis Margins

To set the lower margin for the axis:

`public void setLowerMargin(double margin);`
Sets the lower margin for the axis and sends an `AxisChangeEvent` to all registered listeners. The margin is a percentage of the axis length (for example, `0.05` for a five percent margin).

To set the upper margin for the axis:

`public void setUpperMargin(double margin);`
Sets the upper margin for the axis and sends an `AxisChangeEvent` to all registered listeners. The margin is a percentage of the axis length (for example, `0.05` for a five percent margin).

To set the margin between categories:

`public void setCategoryMargin(double margin);`
Sets the category margin for the axis and sends an `AxisChangeEvent` to all registered listeners. The margin is a percentage of the axis length (for example, `0.20` for a twenty percent margin). The overall margin is distributed over *N-1* gaps where *N* is the number of categories displayed on the axis.

23.9.5 Category Label Positions

To set the position of the category labels:

`public void setCategoryLabelPositions(CategoryLabelPositions positions);`
Sets the attribute that controls the position, alignment and rotation of the category labels along the axis.

The `CategoryLabelPositions` class is just a structure containing four instances of the `CategoryLabelPosition` class. When the axis needs to determine where it is going to draw the category labels, it will select one of those instances depending on the current location of the axis (at the top, bottom, left or right of the plot). It is the attributes of the `CategoryLabelPosition` object that ultimately determine where the labels are drawn.

- the first attribute is an anchor point relative to a notional category rectangle that is computed by the axis (see figure 23.4).Within this rectangle, an *anchor point* is specified using the `RectangleAnchor` class.

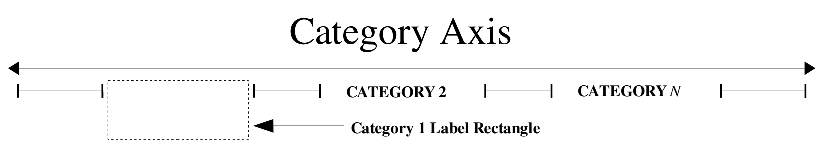


Figure 23.4: A category label rectangle

- the second attribute is a text anchor, which defines a point on the category label which is aligned with the anchor point on within the category rectangle mentioned previously. This is specified using the `TextBlockAnchor` class. Try running the `DrawStringDemo` class in the JFreeChart distribution to see how the anchor is used to align text to a point on the screen.

- the final two attributes are a rotation anchor point and a rotation angle. These are applied once the label text has been positioned using the previous two attributes.

23.9.6 Category Label Tool Tips

It is possible to specify tooltips for the labels along the category axis. This can be useful if you want to use short category names, but have the opportunity to display a longer description. To add a tool tip:

```
public void addCategoryLabelToolTip(Comparable category, String tooltip);
```
Adds a tooltip for the specified category.

To remove a tool tip:

```
public void removeCategoryLabelToolTip(Comparable category);
```
Removes the tooltip for the specified category.

To remove all tool tips:

```
public void clearCategoryLabelToolTips();
```
Removes all category label tool tips.

This feature is not supported by other axis types yet.

23.9.7 Other Methods

To control whether or not a line is drawn for the axis:

```
public void setAxisLineVisible(boolean visible);
```
Sets the flag that controls whether or not a line is drawn for the axis. Often, this isn't required because the `CategoryPlot` draws an outline around itself by default. However, sometimes the plot will have no outline OR the axis may be offset from the plot.

23.9.8 Internals

In JFreeChart, axes are owned/managed by a plot. The plot is responsible for assigning drawing space to all of the axes in a plot, which it does by first asking the axes to estimate the space they require (primarily for the axis labels). The following method is used:

```
public AxisSpace reserveSpace(Graphics2D g2, Plot plot, Rectangle2D plotArea,
RectangleEdge edge, AxisSpace space);
```
Updates the axis space to allow room for this axis to be drawn.

When reserving space, the axis needs to determine the tick marks along the axis, which it does via the following method:

> `public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);`
> Returns a list of the ticks along the axis.

After the plot has estimated the space required for each axis, it then computes the "data area" and draws all the axes around the edges of this area:

> `public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);`
> Draws the axis along a specific edge of the data area. The cursor is a measure of how far from the edge of the data area the axis should be drawn (another axis may have been drawn along the same edge already, for example) and the plot area is the region inside which all the axes should fit (it contains the data area).

For a given rectangular region in Java2D space, the axis can be used to calculate an x-coordinate or a y-coordinate (depending on which edge of the rectangle the axis is aligned) for the start, middle or end of a particular category on the axis:

> `public double getCategoryJava2DCoordinate(CategoryAnchor anchor, int category, int categoryCount, Rectangle2D area, RectangleEdge edge);`
> Returns the x- or y-coordinate (in Java2D space) of the specified category.

23.9.9 Cloning and Serialization

This class is `Cloneable` and `Serializable`.

23.9.10 Notes

Some points to note:

- tick marks are not supported by this axis (yet).

23.10 CategoryAxis3D

23.10.1 Overview

An extension of the `CategoryAxis` class that adds a 3D effect. If you use a `CategoryItemRenderer` that draws items with a 3D effect, then you need to ensure that you are using this class rather than a regular `CategoryAxis`. Eventually, the aim is to combine this class into the `CategoryAxis` class.

23.11 CategoryLabelPosition

23.11.1 Overview

This class records the four attributes that control the position, alignment and rotation of category labels along a `CategoryAxis`.

- the *category anchor* - a `RectangleAnchor` that is used to determine the point on the axis against which the category label is aligned. This is specified relative to a rectangular area that the `CategoryAxis` allocates for the category;
- the *label anchor* - a `TextBlockAnchor` that determines the point on the category label (a `TextBlock`) that is aligned with the category anchor;
- the *rotation anchor* - the point on the category label about which the label is rotated (there may be no rotation);
- the *rotation angle* - the angle of the rotation, specified in radians.

23.11.2 Notes

The following points should be noted:

- instances of this class are immutable, a fact that is relied upon by code elsewhere in the JFreeChart library.

23.12 CategoryLabelPositions

23.12.1 Overview

This class is used to specify the positions of category labels on a `CategoryAxis`. To account for the fact that an axis can appear in one of four different locations (the top, bottom, left or right of the plot) this class contains four instances of the `CategoryLabelPosition` class—the axis will choose the appropriate one when the labels are being drawn.

23.12.2 Usage

For example, to change the category axis labels to a 45 degree angle:

```
CategoryAxis domainAxis = plot.getDomainAxis();
domainAxis.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
```

The above example makes use of the fact that several static instances of this class have been predefined in order to simplify general usage of the `CategoryAxis` class:

Value:	Description:
`STANDARD`	The default label positions.
`UP_90`	The labels are rotated 90 degrees, with the text running from the bottom to the top of the chart.
`DOWN_90`	The labels are rotated 90 degrees, with the text running from the top to the bottom of the chart.
`UP_45`	The labels are rotated 45 degrees, with the text running towards the top of the chart.
`DOWN_45`	The labels are rotated 45 degrees, with the text running towards the bottom of the chart.

Table 23.4: Static instances of the `CategoryLabelPositions` class

However, you can also experiment with creating your own instances of this class, to fully customise the category label positions.

23.13 CategoryTick

23.13.1 Overview

A class used to represent a single tick on a `CategoryAxis`. This class is used internally and it is unlikely that you should ever need to use it directly.

23.14 ColorBar

23.14.1 Overview

A *color bar* is used with a `ContourPlot`.

23.15 CompassFormat

23.15.1 Overview

A custom `NumberFormat` class that can be used to display numerical values as compass directions—see figure 23.5 for an example. In the example, the range

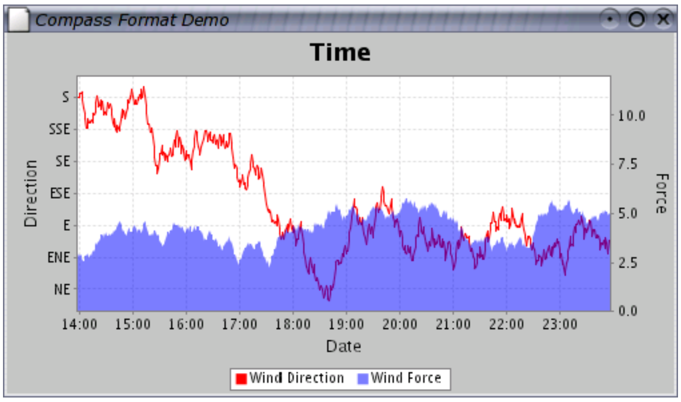


Figure 23.5: A chart that uses the `CompassFormat` class

axis on the left side of the chart displays compass directions in place of numerical values.

23.15.2 Usage

There is a demo (`CompassFormatDemo.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

23.16 DateAxis

23.16.1 Overview

An axis that displays 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.

A date axis can be used for the domain and/or range axis in an `XYPlot`. In a `CategoryPlot`, a date axis can only be used for the range axis.

23.16.2 Constructors

To create a new axis:

> `public DateAxis(String label);`
> Creates a new axis with the specified label (`null` permitted).

23.16.3 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.
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 23.33.3 for information about the attributes inherited by this class.

23.16.4 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 23.16.5;
- the size and formatting of the tick labels, see section 23.16.6;
- other inherited attributes, see section 23.33.4.

23.16.5 The Axis Range

To set the axis range:[1]

```
// start and end are instances of java.util.Date
axis.setRange(start, end);
```

23.16.6 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:

```
// 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);
```

23.16.7 Tick Label Orientation

To control the orientation of the tick labels on the axis:

```
axis.setVerticalTickLabels(true);
```

This code survives from the HorizontalDateAxis class...it needs to be changed to be more generic for axes that could have either a horizontal or vertical orientation.

[1] 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.

23.16.8 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`).

23.17 DateTickMarkPosition

23.17.1 Overview

A simple enumeration of the possible tick mark positions for a `DateAxis`. The positions are:

- `DateTickMarkPosition.START`;
- `DateTickMarkPosition.MIDDLE`;
- `DateTickMarkPosition.END`.

Use the `setTickMarkPosition()` method in the `DateAxis` class to change this setting.

23.18 DateTick

23.18.1 Overview

A class used to represent a single tick on a `DateAxis`.

23.18.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

23.19 DateTickUnit

23.19.1 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.

23.19.2 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.

23.19.3 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 23.19.1. The `count` argument specifies the multiplier (often just `1`).

23.19.4 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 23.19.1.

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.

23.19.5 Notes

This class is immutable, a requirement for all subclasses of TickUnit.

See Also

NumberTickUnit.

23.20 LogarithmicAxis

23.20.1 Overview

A numerical axis that displays values using a logarithmic scale. Extends NumberAxis.

23.21 MarkerAxisBand

23.21.1 Overview

A band that can be added to a NumberAxis to highlight certain value ranges.

23.21.2 Usage

To use this class, first create a new band:

```
MarkerAxisBand band = new MarkerAxisBand(
    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);
```

23.22 NumberAxis

23.22.1 Overview

An axis that displays numerical data along a linear scale. This class extends ValueAxis. You can create your own subclasses if you have special requirements.

23.22.2 Constructors

To create a new axis:

> `public NumberAxis(String label);`
> Creates a new axis with the specified label (`null` permitted).

23.22.3 Usage

A `NumberAxis` 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.

23.22.4 The Axis Range

You can control most aspects of the axis range using methods inherited from the `ValueAxis` class—see section 23.33.5 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.

23.22.5 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.

23.22.6 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.
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 `MarkerAxisBand`).

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"));`

23.22.7 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.

23.22.8 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`.

23.23 NumberAxis3D

23.23.1 Overview

An extension of the `NumberAxis` class that adds a 3D effect. Eventually, this class will be combined with the `NumberAxis` class.

23.24 NumberTick

23.24.1 Overview

A class used to represent a single tick on a `NumberAxis`.

23.24.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

23.25 NumberTickUnit

23.25.1 Overview

A number tick unit for use by subclasses of `NumberAxis` (extends the `TickUnit` class).

23.25.2 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).

23.25.3 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.

23.25.4 Methods

To format a value using the tick unit's internal formatter:

> `public String valueToString(double value);`
> Formats the value as a `String`.

23.25.5 Notes

This class is immutable, a requirement for all subclasses of `TickUnit`.

See Also

`DateTickUnit`.

23.26 SegmentedTimeline

23.26.1 Overview

A segmented timeline for use with a `DateAxis`.

23.26.2 Usage

The `SegmentedHighLowChartDemo` class (included in the JFreeChart distribution) provides an example of how to use this class.

23.27 SymbolicAxis

23.27.1 Overview

An axis that displays numerical data using symbols.

23.28 SymbolicTickUnit

23.28.1 Overview

Not yet documented.

23.29 Tick

23.29.1 Overview

A utility class representing a tick on an axis. Used temporarily during the drawing process only—you won't normally use this class yourself.

See Also

`TickUnit`.

23.30 TickUnit

23.30.1 Overview

An abstract class representing a tick unit, with subclasses including:

- `DateTickUnit` – for use with a `DateAxis`;
- `NumberTickUnit` – for use with a `NumberAxis`.

23.30.2 Constructors

The standard constructor:

```
public TickUnit(double size);
```

Creates a new tick unit with the specified size.

23.30.3 Notes

Implements the `Comparable` interface, so that a collection of tick units can be sorted easily using standard Java methods.

See Also

`TickUnits`.

23.31 TickUnits

23.31.1 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.

23.31.2 Constructors

The default constructor:

> `public TickUnits();`
> Creates a new collection of tick units, initially empty.

23.31.3 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.

23.31.4 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`.

23.32 Timeline

23.32.1 Overview

The interface that defines the methods for a timeline that can be used with a `DateAxis`.

23.32.2 Notes

The `SegmentedTimeline` class implements this interface.

23.33 ValueAxis

23.33.1 Overview

The base class for all axes that display "values", with the two key subclasses being `NumberAxis` and `DateAxis`.

At the lowest level, the axis values are manipulated as `double` primitives, obtained from the `Number` objects supplied by the plot's dataset.

23.33.2 Constructors

The constructors for this class are protected, you cannot create a `ValueAxis` directly—you must use a subclass.

23.33.3 Attributes

The attributes maintained by this class, in addition to those that it inherits from the `Axis` class, are listed in Table 23.5. 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 23.5: Attributes for the `ValueAxis` class

The default values used to initialise the axis attributes (when necessary) are listed in Table 23.6.

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 23.6: `ValueAxis` class default attribute values

23.33.4 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 23.33.5;

23.33.5 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);
```

23.33.6 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 valueToJava2D(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 java2DToValue(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 setLowerBound(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 setUpperBound(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.

23.33.7 Notes

Some points to note:

- 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`.

Chapter 24

Package: org.jfree.chart.entity

24.1 Introduction

The `org.jfree.chart.entity` package contains classes that represent entities in a chart.

24.2 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.

24.3 CategoryItemEntity

24.3.1 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 and URL text (if any) generated for the item, the dataset, and the series and category that the item represents.

24.3.2 Constructors

To construct a new instance:

> `public CategoryItemEntity(Shape area, String toolTipText, String urlText, CategoryDataset dataset, int series, Object category, int categoryIndex);`
> Creates a new entity instance.

24.3.3 Methods

Accessor methods are implemented for the `dataset`, `series` and `category` attributes. Other methods are inherited from the `ChartEntity` class.

24.3.4 Notes

Most `CategoryItemRenderer` implementations will generate entities using this class, as required.

See Also

`ChartEntity`, `CategoryPlot`.

24.4 ChartEntity

24.4.1 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.

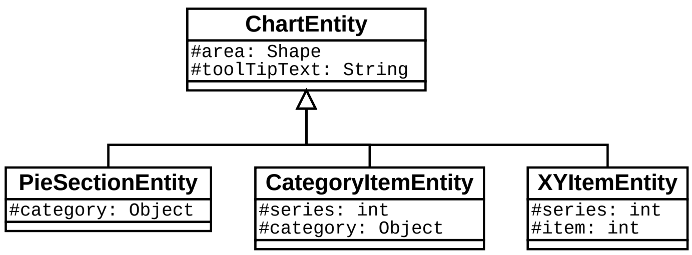


Figure 24.1: Chart entity classes

24.4.2 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.

24.4.3 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.

24.4.4 Notes

The `ChartEntity` class *records* where an entity has been drawn using a `Graphics2D` instance. Changing the attributes of an entity won't change what has already been drawn.

See Also

`CategoryItemEntity`, `PieSectionEntity`, `XYItemEntity`.

24.5 ContourEntity

24.5.1 Overview

Not yet documented.

24.6 EntityCollection

24.6.1 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).

24.6.2 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.

24.6.3 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`.

24.7 PieSectionEntity

24.7.1 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 dataset, pie and section indices, and the tool tip and URL text (if any) generated for the item.

24.7.2 Constructors

To construct a new instance:

> `public PieSectionEntity(Shape area, PieDataset dataset, int pieIndex, int sectionIndex, Comparable sectionKey, String toolTipText, String urlText);`
> Creates a new entity object.

24.7.3 Methods

Accessor methods are implemented for the `dataset`, `pieIndex`, `sectionIndex` and `sectionKey` attributes. Other methods are inherited from the `ChartEntity` class.

24.7.4 Notes

The `PiePlot` class generates pie section entities as required.

See Also

`ChartEntity`, `PiePlot`.

24.8 StandardEntityCollection

24.8.1 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.

24.8.2 Methods

This class implements the methods in the `EntityCollection` interface.

24.8.3 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.[1]

See Also

`ChartEntity`, `EntityCollection`.

24.9 XYItemEntity

24.9.1 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.

24.9.2 Constructors

To construct a new instance:

```
public XYItemEntity(Shape area, XYDataset dataset, int series, int item,
String toolTipText, String urlText);
```

Creates a new entity object.

24.9.3 Methods

Accessor methods are implemented for the `dataset`, `series` and `item` attributes. Other methods are inherited from the `ChartEntity` class.

24.9.4 Notes

Most `XYItemRenderer` implementations will generate entities using this class, as required.

See Also

`ChartEntity`, `XYPlot`.

[1] 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.

Chapter 25

Package: org.jfree.chart.event

25.1 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.

25.2 AxisChangeEvent

25.2.1 Overview

An event that can be sent to an `AxisChangeListener` to provide information about a change to an axis.

25.2.2 Notes

Often, the only information provided by the event is that *some* change has been made to the axis (that is, the specific change is not identified).

25.3 AxisChangeListener

25.3.1 Overview

An interface through which axis change event notifications are posted.

25.3.2 Methods

The interface defines a single method:

```
public void axisChanged(AxisChangeEvent event);
```

Receives notification of a change to an axis.

25.3.3 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.

25.4 ChartChangeEvent

25.4.1 Overview

An event that is used to provide information about changes to a chart. You can register an object with a `JFreeChart` instance, provided that the object implements the `ChartChangeListener` interface, and it will receive a notification whenever the chart changes.

25.4.2 Notes

The `ChartPanel` class automatically registers itself with the chart it is displaying. When it receives a `ChartChangeEvent`, it repaints the chart.

25.5 ChartChangeListener

25.5.1 Overview

An interface through which chart change event notifications are posted.

25.5.2 Methods

The interface defines a single method:

```
public void chartChanged(ChartChangeEvent event);
```

Receives notification of a change to a chart.

25.5.3 Notes

Some points to note:

- 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;
- the `ChartPanel` class implements this interface.

25.6 ChartProgressEvent

25.6.1 Overview

Not yet documented.

25.7 ChartProgressListener

25.7.1 Overview

Not yet documented.

25.8 LegendChangeEvent

25.8.1 Overview

An event that is used to provide information about changes to a legend.

See Also

`LegendChangeListener`.

25.9 LegendChangeListener

25.9.1 Overview

An interface through which legend change event notifications are posted.

25.9.2 Methods

The interface defines a single method:

```
public void legendChanged(LegendChangeEvent event);
```

Receives notification of a change to a legend.

25.9.3 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`.

25.10 PlotChangeEvent

25.10.1 Overview

An event that is used to provide information about changes to a plot. You can register an object with a `Plot` instance, provided that the object implements the `PlotChangeListener` interface, and it will receive a notification whenever the plot changes.

25.10.2 Notes

A `JFreeChart` object will automatically register itself with the `Plot` that it manages, and receive notification whenever the plot changes. The chart usually responds by raising a `ChartChangeEvent`, which other listeners may respond to (for example, the `ChartPanel` if the chart is displayed in a GUI).

25.11 PlotChangeListener

25.11.1 Overview

An interface through which plot change event notifications are posted.

25.11.2 Methods

The interface defines a single method:

> `public void plotChanged(PlotChangeEvent event);`
> Receives notification of a change to a plot.

25.11.3 Notes

Some points to note:

- 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.
- the `JFreeChart` class implements this interface and automatically registers itself with the plot it manages.

25.12 RendererChangeEvent

25.12.1 Overview

An event that is used to provide information about changes to a renderer. If an object needs to receive notification of these events, its class should implement the `RendererChangeListener` interface so the object can register itself with the renderer via the `addChangeListener()` method.

In the default setup, a change to a renderer will cause the plot to receive notification of the event. The plot will usually respond by firing a `PlotChangeEvent` (which usually gets passed on to the chart and results in a `ChartChangeEvent` being fired).

25.12.2 Notes

In the current implementation, the event just signals a change without specifying exactly what changed. A possible future enhancement would be to include information about the nature of the change, so that the listener(s) can decide what action to take in response to the event.

25.13 RendererChangeListener

25.13.1 Overview

An interface through which renderer change event notifications are posted. The `CategoryPlot` and `XYPlot` classes implement this interface so they can receive notification of changes to their renderer(s).

25.13.2 Methods

The interface defines a single method:

```
public void rendererChanged(RendererChangeEvent event);
```

Receives notification of a change to a renderer.

25.13.3 Notes

If an `Object` needs to receive notification of changes to a renderer, then its class needs to implement this interface so the object can register itself with the renderer.

25.14 TitleChangeEvent

25.14.1 Overview

An event that is used to provide information about changes to a chart title (any subclass of `Title`).

25.14.2 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

`Title`, `TitleChangeListener`.

25.15 TitleChangeListener

25.15.1 Overview

An interface through which title change event notifications are posted.

25.15.2 Methods

The interface defines a single method:

```
public void titleChanged(TitleChangeEvent event);
```

Receives notification of a change to a title.

25.15.3 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`.

Chapter 26

Package: org.jfree.chart.imagemap

26.1 Overview

This package contains classes and interfaces that support the creation of HTML image maps. These image maps can be created using the `ChartUtilities` class, typically from a servlet.

26.2 DynamicDriveToolTipTagFragmentGenerator

26.2.1 Overview

A tool-tip fragment generator that generates tool-tips that are designed to work with the Dynamic Drive DHTML Tip Message library:

`http://www.dynamicdrive.com`

This class implements the `ToolTipTagFragmentGenerator` interface.

26.3 OverLIBToolTipTagFragmentGenerator

26.3.1 Overview

A tool-tip generator that generates tool-tips for use with the OverLIB library. See this URL for details:

`http://www.bosrup.com/web/overlib/`

This class implements the `ToolTipTagFragmentGenerator` interface.

26.4 StandardToolTipTagFragmentGenerator

26.4.1 Overview

A tool-tip generator that generates tool-tips using the HTML title attribute.

This class implements the `ToolTipTagFragmentGenerator` interface.

26.5 StandardURLTagFragmentGenerator

26.5.1 Overview

A standard implementation of the `URLTagFragmentGenerator` interface.

26.6 ToolTipTagFragmentGenerator

26.6.1 Overview

The interface that must be implemented by a class that generates tooltip tag fragments for an HTML image map.

Classes that implement this interface include:

- `StandardToolTipTagFragmentGenerator`;
- `DynamicDriveToolTipTagFragmentGenerator`;
- `OverLIBToolTipTagFragmentGenerator`;

26.6.2 Methods

This interface defines a single method:

```
public String generateToolTipFragment(String toolTipText);
```

Returns a tooltip fragment based on the supplied tool-tip text.

26.7 URLTagFragmentGenerator

26.7.1 Overview

The interface that must be implemented by a class that generates URL tag fragments for an HTML image map.

The `StandardURLTagFragmentGenerator` class provides one implementation of this interface.

26.7.2 Methods

This interface defines a single method:

```
public String generateURLFragment(String urlText);
```

Returns a URL fragment based on the supplied URL text.

Chapter 27

Package: org.jfree.chart.labels

27.1 Introduction

This package contains interfaces and classes for generating labels for the individual data items in a chart. There are two label types:

- *item labels* – small text items displayed as part of the chart;
- *tooltips* – text that is displayed when the mouse pointer "hovers" over an item.

Section 10 contains information about using tool tips with JFreeChart.

27.2 BoxAndWhiskerItemLabelGenerator

27.2.1 Overview

A *label generator* for a box-and-whisker chart. This is the default generator used by the `XYBoxAndWhiskerRenderer` class.

27.3 CategoryItemLabelGenerator

27.3.1 Overview

A *category item label generator* is an object that assumes responsibility for creating the text strings that will be used for item labels in a chart. A generator is assigned to a renderer using the `setLabelGenerator()` method in the `CategoryItemRenderer` interface. This interface defines the API through which the renderer will communicate with the generator.

27.3.2 Methods

The renderer will call this method to obtain an item label:

> `public String generateItemLabel(CategoryDataset data,`
> `int series, int category);`
> Returns a string that will be used to label the specified item. Classes that implement this method are permitted to return `null` for the result.

27.3.3 Notes

Some points to note:

- the `StandardCategoryItemLabelGenerator` provides one implementation of this interface, but you can also write your own class that implements this interface, and take complete control over the generation of item labels;
- refer to chapter 11 for information about using item labels.

27.4 CategoryToolTipGenerator

27.4.1 Overview

A *category tool tip generator* is an object that assumes responsibility for creating the text strings that will be used for tooltips in a chart. A generator is assigned to a renderer using the `setToolTipGenerator()` method in the `CategoryItemRenderer` interface. This interface defines the API through which the renderer will communicate with the generator.

27.4.2 Methods

The renderer will call this method to obtain the tooltip text for an item:

> `public String generateToolTip(CategoryDataset data,`
> `int series, int category);`
> Returns a string that will be used as the tooltip text for the specified item. If `null` is returned, no tool tip will be displayed.

27.4.3 Notes

Some points to note:

- the `StandardCategoryItemLabelGenerator` provides one implementation of this interface, but you can also write your own class that implements this interface, and take complete control over the generation of item labels and tooltips;
- refer to chapter 10 for information about using tool tips.

27.5 ContourToolTipGenerator

27.5.1 Overview

The interface that must be implemented by all contour tool tip generators. When a `ContourPlot` requires tooltip text for a data item, it will obtain it via this interface.

27.5.2 Methods

The interface defines a single method for obtaining the tooltip text for a data item:

> `public String generateToolTip(ContourDataset data, int item);`
> Returns a string that can be used as the tooltip text for a data item.

27.6 CustomXYItemLabelGenerator

27.6.1 Overview

A tool tip generator (for use with an `XYItemRenderer`) that returns a predefined tool tip for each data item.

27.6.2 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.

27.6.3 Notes

See section 10 for information about using tool tips with JFreeChart.

27.7 HighLowItemLabelGenerator

27.7.1 Overview

A *tooltip generator* that is intended for use with the `HighLowRenderer` class. The generator will only return tooltips for a dataset that is an implementation of the `HighLowDataset` interface.

27.7.2 Methods

The key method constructs a `String` to be used as the tooltip text for a particular data item:

> `public String generateToolTip(XYDataset dataset, int series, int item);`
> Returns a string containing the date, value, high value, low value, open value and close value for the data item. This method will return `null` if the dataset does not implement the `HighLowDataset` interface.

27.7.3 Notes

See section 10 for an overview of tool tips with JFreeChart.

27.8 IntervalCategoryItemLabelGenerator

27.8.1 Overview

An *item label generator* that can be used with any `CategoryItemRenderer`. This generator will detect if the dataset supplied to the renderer is an implementation of the `IntervalCategoryDataset` interface, and will generate labels that display both the *start value* and the *end value* for each item.

27.8.2 Constructors

The default constructor will create a label generator that formats the data values as numbers, using the platform default number format:

> `public IntervalCategoryItemLabelGenerator();`
> Creates a new label generator with a default number formatter.

If you prefer to set the number format yourself, use the following constructor:

> `public IntervalCategoryItemLabelGenerator(NumberFormat formatter);`
> Creates a new label generator with a specific number formatter.

In some cases, the data values in the dataset will represent dates (encoded as milliseconds since midnight, 1-Jan-1970 GMT, as for `java.util.Date`). In this case, you can create a label generator using the following constructor:

> `public IntervalCategoryItemLabelGenerator(DateFormat formatter);`
> Creates a new label generator that formats the start and end data values as dates.

27.8.3 Notes

The `createGanttChart()` in the `ChartFactory` class uses this type of label generator (with date formatting).

27.9 ItemLabelAnchor

27.9.1 Overview

An *item label anchor* is used by a renderer to calculate a fixed point (the *item label anchor point*) relative to a data item on a chart. This point becomes a reference point that an item label can be aligned to.

This class defines 25 anchors. The numbers 1 to 12 are used and roughly correspond to the positions of the hours on a clock face. In addition, positions are defined relative to an “inside” ring and an “outside” ring - see figure 27.1 for an illustration.
With 12 points on the inside circle, 12 points on the outside circle, plus a “center” anchor point, in all there are 25 possible anchor points.

For some renderers, the circular arrangement of anchor points doesn’t make sense, so the renderer is free to modify the anchor positions (see the `BarRenderer` class for an example).

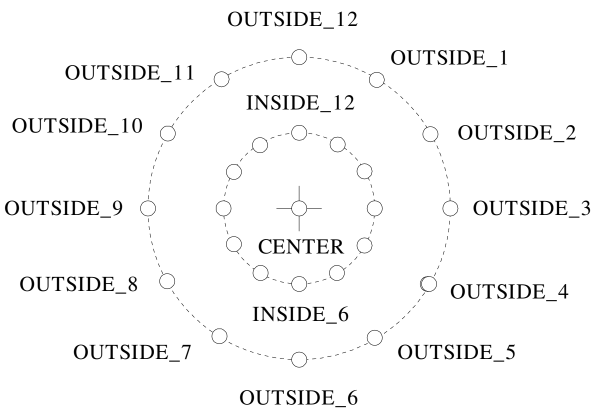


Figure 27.1: The Item Label Anchors

27.9.2 Usage

The `ItemLabelPosition` class includes an item label anchor as one of the attributes that define the location of item labels drawn by a renderer.

27.10 ItemLabelPosition

27.10.1 Overview

This class is used to specify the position of item labels on a chart. Four attributes are used to specify the position:

- the *item label anchor* - the renderer will use this to calculate an *(*x, y) anchor point on the chart near to the data item that the item label corresponds to (see `ItemLabelAnchor`);
- the *text anchor* - this is a point relative to the item label text which will be aligned with the item label anchor point above;
- the *rotation anchor* - this is another point somewhere on the item label about which the text will be rotated (if there is a rotation);
- the *rotation angle* - this specifies the amount of rotation about the rotation point.

These four attributes provide a lot of scope for placing item labels in interesting ways.

27.10.2 Usage

The `AbstractRenderer` class provides methods for specifying the item label position for positive and negative data values separately:

> `public void setPositiveItemLabelPosition(ItemLabelPosition position);`
> Sets the item label position for positive data values.

> `public void setNegativeItemLabelPosition(ItemLabelPosition position);`
> Sets the item label position for negative data values.

27.11 PieSectionLabelGenerator

27.11.1 Overview

The interface that must be implemented by a *pie section label generator*, a class used to generate section labels for a pie chart.

27.11.2 Methods

The `PiePlot` class will call the following method to obtain a section label for each section in a pie chart as it is being drawn:

> `public String generateSectionLabel(PieDataset dataset, Comparable key);`
> Returns a section label for the specified item in the dataset.

27.11.3 Notes

Some points to note:

- the `StandardPieItemLabelGenerator` class provides an implementation of this interface;
- you can develop your own label generator, register it with a `PiePlot`, and take full control over the labels that are generated.

27.12 PieToolTipGenerator

27.12.1 Overview

The interface that must be implemented by a *pie tool tip generator*, a class used to generate tool tips for a pie chart.

27.12.2 Methods

The `PiePlot` class will call the following method to obtain a tooltip for each section in a pie chart:

> `public String generateToolTip(PieDataset data, Comparable key);`
> Returns a `String` that will be used as the tool tip text.

27.12.3 Notes

Some points to note:

- the StandardPieItemLabelGenerator class provides an implementation of this interface;
- you can develop your own tool tip generator, register it with a PiePlot, and take full control over the labels that are generated;
- section 10 contains information about using tool tips with JFreeChart.

27.13 StandardCategoryItemLabelGenerator

27.13.1 Overview

A generator that can be assigned to a CategoryItemRenderer for the purpose of generating item labels and/or tooltips. This class implements the following interfaces:

- CategoryItemLabelGenerator
- CategoryToolTipGenerator.

There is a plan to rewrite this class to make use of Java's MessageFormat class. This will provide greater flexibility in the label format without having to subclass.

27.13.2 Usage

Most often you will assign a generator to a renderer and then never need to refer to it again:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryItemRenderer renderer = chart.getPlot().getRenderer();
renderer.setItemLabelGenerator(new StandardCategoryItemLabelGenerator());
renderer.setToolTipGenerator(new StandardCategoryItemLabelGenerator());
```

The renderer will call the generator's methods when necessary.

27.13.3 Constructors

This class has a default constructor:

> `public StandardCategoryItemLabelGenerator();`
> Creates a new generator that formats values using the default number format for the user's locale.

To create a generator that formats values as numbers:

> `public StandardCategoryItemLabelGenerator(NumberFormat formatter);`
> Creates a generator that formats values using the supplied formatter.

`public StandardCategoryItemLabelGenerator(NumberFormat formatter, boolean showSeriesNameInToolTips);`
Creates a generator that formats values using the supplied formatter and sets a flag that controls whether or not the series name is displayed in the tooltips generated.

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 StandardCategoryItemLabelGenerator(DateFormat formatter);`
Creates a generator that formats values as dates using the supplied formatter.

`public StandardCategoryItemLabelGenerator(DateFormat formatter, boolean showSeriesNamesInToolTips);`
Creates a generator that formats values as dates using the supplied formatter and sets a flag that controls whether or not the series name is displayed in the tooltips generated.

27.13.4 Methods

The following methods are typically called by the renderer, you won't need to call them directly yourself.

To generate a tooltip:

`public String generateToolTip(CategoryDataset dataset, int series, int category);`
Generates a tooltip for the specified data item. This class generates tooltips in the format `<series-name>, <category-name> = <value>` where the value is formatted as a number or a date, depending on the constructor used to create the generator.

To generate an item label:

`public String generateItemLabel(CategoryDataset dataset, int series, int category);`
Generates an item label for the specified data item. For this generator, the value returned is simply the data value formatted with the formatter supplied in the constructor.

27.13.5 Notes

Some points to note:

- this class implements the `PublicCloneable` interface;
- section 10 contains information about using tool tips with JFreeChart.
- section 11 contains information about using item labels with JFreeChart.

27.14 StandardContourToolTipGenerator

27.14.1 Overview

A default implementation of the `ContourToolTipGenerator` interface.

27.15 StandardPieItemLabelGenerator

27.15.1 Overview

A label generator (implements `PieItemLabelGenerator`) that can be used to generate section labels and tool tips for a `PiePlot`.

This class uses a `MessageFormat` instance to compose labels. Three items are available for use in the labels:

- the item key (as a `String`);
- the item value (converted to a `String` using a `NumberFormat` instance);
- the item percentage (converted to a `String` using a `NumberFormat` instance.

The default tool tip format string is `"{0}: ({1}, {2})"`, which displays the item key, followed by the item value and percentage. Similarly, the default section label format is `"{0} = {1}"`, which displays the item key followed by the item value (the percentage is not displayed).

27.15.2 Constructors

The default constructor uses a number formatter for the default locale:

> `public StandardPieItemLabelGenerator();`
> Creates a default label generator.

You can create a generator with a specific format string:

> `public StandardPieItemLabelGenerator(String labelFormat);`
> Creates a generator using the specified format string. The item value and percentage (if included in the format string) will be formatted using default formatters for the current locale.

The final constructor allows you to specify the item value and percentage formatters:

> `public StandardPieItemLabelGenerator(String labelFormat,`
> `NumberFormat numberFormat, NumberFormat percentFormat)`
> Creates a generator using the specifed format string, with custom formatters for the item value and item percentage.

27.15.3 Notes

Section 10 contains information about using tool tips with JFreeChart.

27.16 StandardXYItemLabelGenerator

27.16.1 Overview

A standard implementation of the `XYItemLabelGenerator` interface. This class generates tool tips in the format:

```
<series-name> : x: <x-value>, y: <y-value>
```

27.16.2 Constructors

To create a tool tip generator:

> `public StandardXYItemLabelGenerator(NumberFormat xFormat, NumberFormat yFormat);`
> Creates a tool tip generator that uses the supplied number formatters for the x and y values.

27.16.3 Notes

Section 10 contains information about using tool tips with JFreeChart.

27.17 StandardXYZItemLabelGenerator

27.17.1 Overview

A default implementation of the `XYZItemLabelGenerator` interface. This generator is used with the `XYBubbleRenderer` class.

27.18 SymbolicXYItemLabelGenerator

27.18.1 Overview

An item label generator for use with symbolic plots.

27.19 XYItemLabelGenerator

27.19.1 Overview

The interface that must be implemented by an *XY item label generator*, a class used to generate item labels and tool tips for an `XYPlot`.

27.19.2 Methods

The plot will call the following method whenever it requires an item label:

> `public String generateItemLabel(XYDataset dataset, int series, int item);`
> Creates an item label for the specified item in the dataset.

The plot will call the following method whenever it requires a tool tip for an item:

`public String generateToolTip(XYDataset data, int series, int item);`
This method is called whenever the plot needs to generate a tooltip for a data item. It can return an arbitrary string, generally derived from the specified item in the supplied dataset.

27.19.3 Notes

Some points to note:

- to "install" a tool tip generator, use the `setToolTipGenerator()` method in the `XYItemRenderer` interface.
- `StandardXYItemLabelGenerator` implements this interface, but you are free to write your own implementation to suit your requirements.

Section 10 contains information about using tool tips with JFreeChart.

27.20 XYToolTipGenerator

27.20.1 Overview

The interface that must be implemented by an *XY tool tip generator*, a class used to generate tool tips for an `XYPlot`.

27.20.2 Methods

The plot will call the following method whenever it requires a tool tip for an item:

`public String generateToolTip(XYDataset data, int series, int item);`
This method is called whenever the plot needs to generate a tooltip for a data item. It can return an arbitrary string, generally derived from the specified item in the supplied dataset.

27.20.3 Notes

Some points to note:

- to "install" a tool tip generator, use the `setToolTipGenerator()` method in the `XYItemRenderer` interface.
- `StandardXYItemLabelGenerator` implements this interface, but you are free to write your own implementation to suit your requirements.

Section 10 contains information about using tool tips with JFreeChart.

27.21 XYZItemLabelGenerator

27.21.1 Overview

A label generator that creates labels for items in an `XYZDataset`.

Chapter 28

Package: org.jfree.chart.needle

28.1 Overview

This package contains classes for drawing needles in a compass plot:

- `ArrowNeedle` – an arrow needle;
- `LineNeedle` – a line needle;
- `LongNeedle` – a long needle;
- `PinNeedle` – a pin needle;
- `PlumNeedle` – a plum needle;
- `PointerNeedle` – a pointer needle;
- `ShipNeedle` – a ship needle;
- `WindNeedle` – a wind needle;

28.2 ArrowNeedle

28.2.1 Overview

A class that draws an *arrow needle* for the `CompassPlot` class (see figure 28.1).

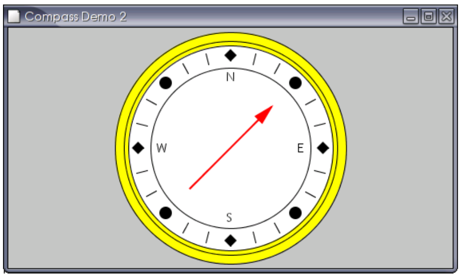


Figure 28.1: An arrow needle

28.3 LineNeedle

28.3.1 Overview

A class that draws a *line needle* for the `CompassPlot` class (see figure 28.2).

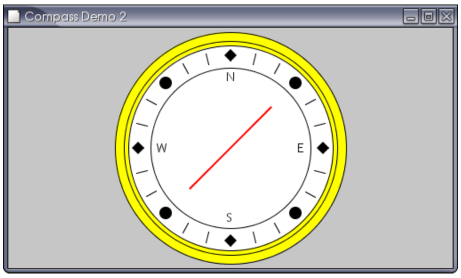


Figure 28.2: A line needle

28.4 LongNeedle

28.4.1 Overview

A class that draws a *long needle* for the `CompassPlot` class (see figure 28.3).

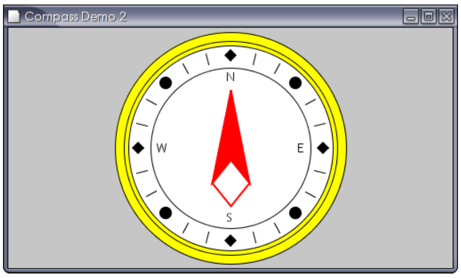


Figure 28.3: A long needle

28.5 MeterNeedle

28.5.1 Overview

A base class that draws a needle for the `CompassPlot` class. A range of different subclasses implement different types of needles:

- `ArrowNeedle` – an arrow needle;
- `LineNeedle` – a line needle;
- `LongNeedle` – a long needle;
- `PinNeedle` – a pin needle;
- `PlumNeedle` – a plum needle;
- `PointerNeedle` – a pointer needle;
- `ShipNeedle` – a ship needle;
- `WindNeedle` – a wind needle;

28.6 PinNeedle

28.6.1 Overview

A class that draws a *pin needle* for the CompassPlot class (see figure 28.4).

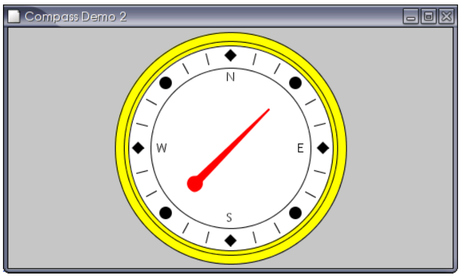


Figure 28.4: A pin needle

28.7 PlumNeedle

28.7.1 Overview

A class that draws a *plum needle* for the CompassPlot class (see figure 28.5).

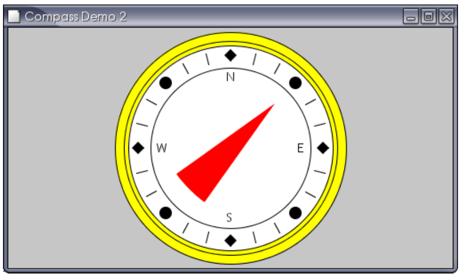


Figure 28.5: A plum needle

28.8 PointerNeedle

28.8.1 Overview

A class that draws a *pointer needle* for the `CompassPlot` class (see figure 28.6).

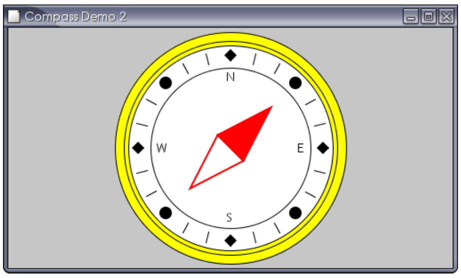


Figure 28.6: A pointer needle

28.9 ShipNeedle

28.9.1 Overview

A class that draws a *ship needle* for the `CompassPlot` class (see figure 28.7).

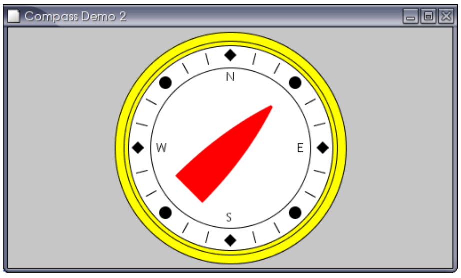


Figure 28.7: A ship needle

28.10 WindNeedle

28.10.1 Overview

A class that draws a *wind needle* for the CompassPlot class (see figure 28.8).

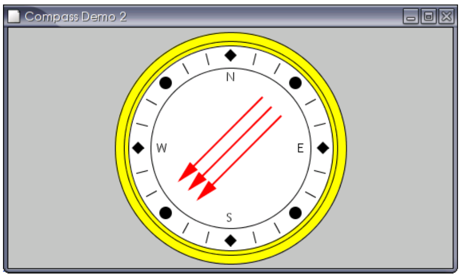


Figure 28.8: A wind needle

Chapter 29

Package: org.jfree.chart.plot

29.1 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.

29.2 CategoryPlot

29.2.1 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. A *category plot* has:

- a *domain axis* (a `CategoryAxis`);
- a *range axis* (a `ValueAxis`);
- a *dataset* (any instance of the `CategoryDataset` interface);
- a *renderer* (any instance of the `CategoryItemRenderer` interface);
- any number of *secondary datasets*, *secondary range axes* and *secondary renderers*;

The plot can be displayed with a horizontal or vertical orientation.

29.2.2 Attributes

The attributes maintained by the `CategoryPlot` class, which are in addition to those inherited from the `Plot` class, are listed in Table 29.1.

Attribute:	Description:
orientation	The plot orientation (horizontal or vertical).
axisOffset	The offset between the data area and the axes.
domainAxis	The domain axis (used to display categories).
domainAxisLocation	The location of the domain axis.
secondaryDomainAxes	A list of (optional) secondary domain axes.
secondaryDomainAxisLocations	A list of secondary domain axis locations.
rangeAxis	The range axis (used to display values).
rangeAxisLocation	The location of the range axis.
secondaryRangeAxes	A list of (optional) secondary range axes.
secondaryRangeAxisLocations	A list of secondary range axis locations.
dataset	The dataset.
secondaryDatasets	A list of (optional) secondary datasets.
renderer	The plot's renderer (a "pluggable" object responsible for drawing individual data items within the plot).
secondaryRenderers	A list of (optional) secondary renderers.
renderingOrder	The order for rendering data items.
domainGridlinesVisible	A flag that controls whether gridlines are drawn against the domain axis.
domainGridlinePosition	The position of the gridlines against the domain axis.
domainGridlinePaint	The paint used to draw the domain gridlines.
domainGridlineStroke	The stroke used to draw the domain gridlines.
rangeGridlinesVisible	A flag that controls whether gridlines are drawn against the range axis.
rangeGridlinePaint	The paint used to draw the range gridlines.
rangeGridlineStroke	The stroke used to draw the range gridlines.
rangeMarkers	A list of markers (constants) to be highlighted on the plot.

Table 29.1: Attributes for the `CategoryPlot` class

29.2.3 Axes

The plot's domain axis is an instance of `CategoryAxis`. You can obtain a reference to the axis with:

```
CategoryAxis domainAxis = myPlot.getDomainAxis();
```

The plot's range axis is an instance of `ValueAxis`. You can obtain a reference to the axis with:

```
ValueAxis rangeAxis = myPlot.getRangeAxis();
```

The axis classes have many attributes that can be customised to control the appearance of your charts.

29.2.4 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 and set the series colors using code similar to the following:

```
CategoryPlot plot = myChart.getCategoryPlot();
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setSeriesPaint(0, new Color(0, 0, 255));
renderer.setSeriesPaint(1, new Color(75, 75, 255));
renderer.setSeriesPaint(2, new Color(150, 150, 255));
```

29.2.5 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.

29.2.6 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.

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.

To add a range marker to a plot:

> `public void addRangeMarker(Marker marker);`
> Adds a marker which will be drawn against the range axis.

To add an annotation to a plot:

> `public void addAnnotation(CategoryAnnotation annotation);`
> Adds an annotation to the plot.

To set the weight for a plot:

> `public void setWeight(int weight);`
> Sets the weight for a plot. This is used to determine how much space is allocated to the plot when it is used as a subplot within a combined plot.

29.2.7 Notes

A number of `CategoryItemRenderer` implementations are included in the JFreeChart distribution.

See Also

`CombinedDomainCategoryPlot`, `CombinedRangeCategoryPlot`.

29.3 CombinedDomainCategoryPlot

29.3.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared domain axis.

29.3.2 Notes

The `CombinedCategoryPlotDemo1.java` file (included in the JFreeChart distribution) provides an example of this type of plot.

See Also

`CombinedRangeCategoryPlot`.

29.4 CombinedDomainXYPlot

29.4.1 Overview

A subclass of `XYPlot` that allows you to combined multiple plots on one chart, where the subplots share the domain axis, and maintain their own range axes.

Figure 29.1 illustrates the relationship between the `CombinedDomainXYPlot` and its subplots).

The `CombinedXYPlotDemo1` class (included in the JFreeChart distribution) provides an example of this type of plot.

29.4.2 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 `CombinedDomainXYPlot` can be any instance of `XYPlot` and should have its domain 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

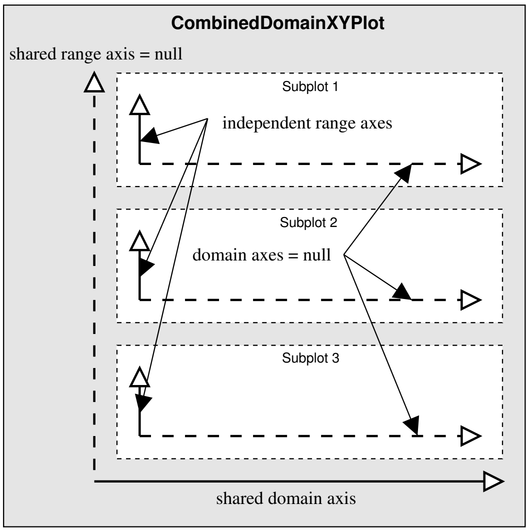


Figure 29.1: CombinedDomainXYPlot axes

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.

29.4.3 Notes

Some points to note:

- 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;
- a demonstration of this type of plot is described in section **??**.

See Also

`XYPlot`.

29.5 CombinedRangeCategoryPlot

29.5.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared range axis.

29.5.2 Notes

The `CombinedCategoryPlotDemo2.java` file (included in the JFreeChart distribution) provides an example of this type of plot.

29.6 CombinedRangeXYPlot

29.6.1 Overview

A subclass of `XYPlot` that allows you to combined multiple plots on one chart, where the subplots share a single range axis, and maintain their own domain axes.

Figure 29.2 illustrates the relationship between the `CombinedRangeXYPlot` and its subplots).

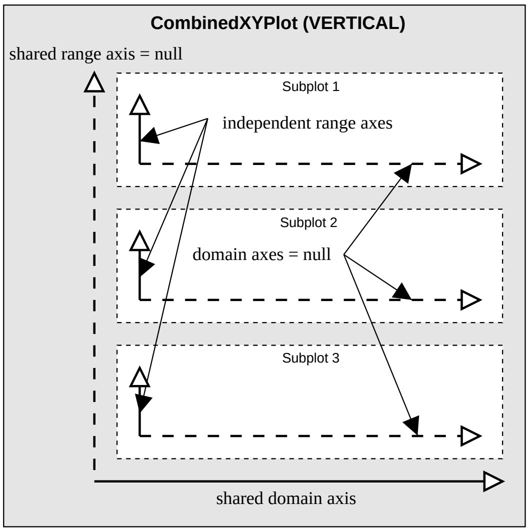


Figure 29.2: CombinedRangeXYPlot axes

The `CombinedRangeXYPlotDemo` class provides an example of this type of plot.

29.6.2 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 `CombinedRangeXYPlot` can be any instance of `XYPlot` 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.

29.6.3 Notes

Some points to note:

- 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 **??**.

29.7 CompassPlot

29.7.1 Overview

A *compass plot* presents directional data in the form of a compass dial.

29.7.2 Notes

There is a demonstration `CompassDemo.java` application included in the JFreeChart distribution.

29.8 ContourPlot

29.8.1 Overview

A custom plot that displays *(x, y, z)* data in the form of a 2D contour plot.

29.9 ContourPlotUtilities

29.9.1 Overview

A class that contains static utility methods used by the contour plot implementation.

29.10 ContourValuePlot

29.10.1 Overview

An interface used by the contour plot implementation.

29.11 CrosshairState

29.11.1 Overview

This class maintains information about the crosshairs on a plot, as the plot is being rendered. Crosshairs will often need to "lock on" to the data point nearest to the anchor point (which is usually set by a mouse click). This class keeps track of the data item that is "closest" (either in screen space or in data space) to the anchor point.

29.11.2 Constructors

The default constructor:

> `public CrosshairState();`
> Creates a new instance where distance is calculated in screen space.

> `public CrosshairState(boolean calculateDistanceInDataSpace);`
> Creates a new instance where you can select to measure distance in data space or screen space.

29.11.3 Methods

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

> `public void updateCrosshairPoint(double candidateX, double candidateY);`
> Considers the candidate point and updates the crosshair point if the candidate is the "closest" to the anchor point.

29.12 DefaultDrawingSupplier

29.12.1 Overview

A default class used to provide a sequence of unique `Paint`, `Stroke` and `Shape` objects to be used by renderers when drawing charts (this class implements the `DrawingSupplier` interface).

29.12.2 Usage

Every `Plot` class is initialised with an instance of this class as its drawing supplier, and it is unlikely that you would need to use this class directly. However, you *might* create your own class that implements the `DrawingSupplier` interface, and register it with the plot, as a way of overriding the default series colors, line styles and shapes.

29.13 DrawingSupplier

29.13.1 Overview

A *drawing supplier* provides a limitless (but ultimately repeating) sequence of `Paint`, `Stroke` and `Shape` objects that can be used by renderers when drawing charts.

All `Plot` classes will have a default drawing supplier. This provides a single source for colors and line styles, which is particularly useful for avoiding duplicates when a plot has multiple renderers.

You can register your own drawing supplier with a plot if you want to modify the default behaviour. If you do this, you need to call the plot's `setDrawingSupplier()` method before the chart is first drawn (the reason being that the plot's renderer(s) will cache the values returned by the drawing supplier the first time a chart is drawn—subsequent changes to the drawing supplier will have no effect on the values already cached).

29.13.2 Methods

To obtain the next `Paint` object in the sequence:

> `public Paint getNextPaint();`
> Returns the next `Paint` object in the sequence (never `null`). These are usually used as the default series colors in charts.

> `public Paint getNextOutlinePaint();`
> Returns the next outline `Paint` object in the sequence (never `null`).

> `public Stroke getNextStroke();`
> Returns the next `Stroke` object in the sequence (never `null`). These are usually used as the default series line style in charts.

> `public Stroke getNextOutlineStroke();`
> Returns the next outline `Stroke` object in the sequence (never `null`).

> `public Shape getNextShape();`
> Returns the next `Shape` object in the sequence (never `null`). The shapes returned by this method should be centered on (0, 0) in Java2D coordinates.

29.14 FastScatterPlot

29.14.1 Overview

A custom plot that aims to be fast rather than flexible. A couple of techniques are used to make this plot type faster than the other plot types provided by JFreeChart:

- data is obtained directly from an array rather than via the `XYDataset` interface;

- the plot draws each point directly rather than using a plug-in renderer.

This class is still at the "proof of concept" stage. It works reasonably well but doesn't provide a lot of options.

29.14.2 Methods

This class overrides the `draw()` method defined in the `Plot` class:

> `public void draw(Graphics2D g2, Rectangle2D plotArea, PlotState parentState, PlotRenderingInfo info);`
> Draws the plot in the specified area. You won't normally call this method directly, it is called for you by the `JFreeChart` class.

29.14.3 Gridlines

You can display gridlines against the *domain axis* using the following methods:

> `public void setDomainGridlinesVisible(boolean visible);`
> Sets a flag that controls whether or not the gridlines are displayed and sends a `PlotChangeEvent` to all registered listeners.

> `public void setDomainGridlinePaint(Paint paint);`
> Sets the `Paint` used for the domain gridlines and sends a `PlotChangeEvent` to all registered listeners.

> `public void setDomainGridlineStroke(Stroke stroke);`
> Sets the `Stroke` used for the domain gridlines and sends a `PlotChangeEvent` to all registered listeners.

Similarly, you can display gridlines against the *range axis*:

> `public void setRangeGridlinesVisible(boolean visible);`
> Sets a flag that controls whether or not the gridlines are displayed and sends a `PlotChangeEvent` to all registered listeners.

> `public void setRangeGridlinePaint(Paint paint);`
> Sets the `Paint` used for the range gridlines and sends a `PlotChangeEvent` to all registered listeners.

> `public void setRangeGridlineStroke(Stroke stroke);`
> Sets the `Stroke` used for the range gridlines and sends a `PlotChangeEvent` to all registered listeners.

29.14.4 Notes

Some points to note:

- this plot does not support secondary axes;
- there is a demo (`FastScatterPlotDemo.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

29.15 IntervalMarker

29.15.1 Overview

An *interval marker* is used to highlight a (fixed) range of values against the domain or range axis for a CategoryPlot or an XYPlot. This class extends the Marker class.

29.15.2 Usage

There is a demo application (DifferenceChartDemo2.java) included in the JFreeChart distribution (in the src/org/jfree/chart/demo directory) that illustrates the use of this class.

29.15.3 Notes

Some points to note:

- this class is Cloneable and Serializable.

29.16 Marker

29.16.1 Overview

The base class for markers that can be added to a CategoryPlot or an XYPlot. There are two subclasses, as listed in Table 29.2.

Class:	Description:
ValueMarker	A marker that highlights a single value.
IntervalMarker	A marker that highlights a range of values.

Table 29.2: Subclasses of Marker

Markers are used to highlight particular values or value ranges against either the domain or range axes. Labels can be added to the markers.

29.16.2 Usage

There is a demo application (MarkerDemo1.java) included in the JFreeChart distribution (in the src/org/jfree/chart/demo directory) that illustrates the use of markers.

29.16.3 Notes

Some points to note:

- markers should be Cloneable and Serializable.

29.17 MeterPlot

29.17.1 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.

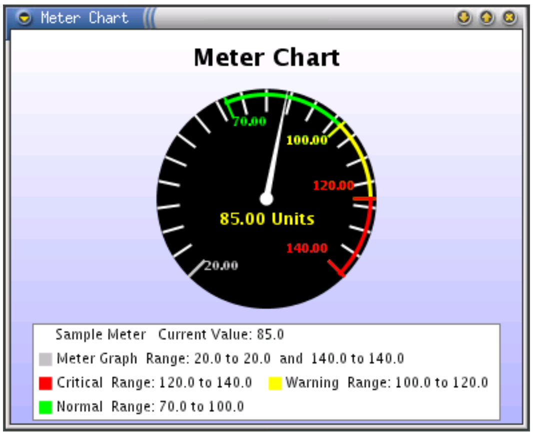


Figure 29.3: A meter chart

Three ranges on the dial provide some context for the value: the *normal range*, the *warning range* and the *critical range*.

29.17.2 Constructors

To create a new `MeterPlot`:

> `public MeterPlot(MeterDataset dataset);`
> 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 dataset, 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.

29.17.3 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 set this to `null`, no background is painted.

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).

29.17.4 Notes

This chart type was contributed by Hari.

The `MeterPlotDemo` class in the `org.jfree.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`.

29.18 MultiplePiePlot

29.18.1 Overview

A specialised plot that displays data from a `CategoryDataset` in the form of multiple pie charts. Figure 29.4 shows an example.

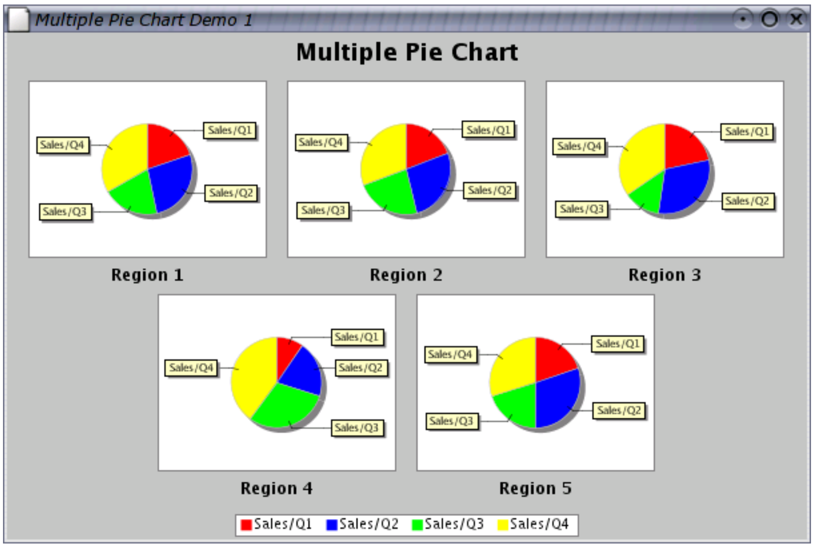


Figure 29.4: A multiple pie chart

29.18.2 Notes

Some points to note:

- a demo application (`MultiplePieChartDemo1.java`) is included in the JFreeChart distribution.
- the `createMultiplePieChart()` and `createMultiplePieChart3D()` methods in the `ChartFactory` class that create charts using this plot.

29.19 PiePlot

29.19.1 Overview

The `PiePlot` class draws pie charts using data obtained through the `PieDataset` interface. A sample chart is shown in figure 29.5. A related class, `PiePlot3D`,

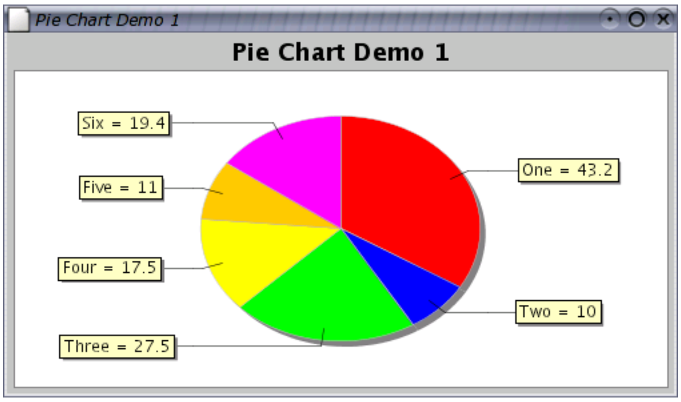


Figure 29.5: A sample pie chart

draws pie charts with a 3D effect.

29.19.2 Constructors

To construct a pie plot:

> `public PiePlot(PieDataset dataset);`
> Creates a pie plot for the given dataset. All plot attributes are initialised with default values—these can be changed at any time.

29.19.3 Attributes

The attributes maintained by the `PiePlot` class, which are in addition to those inherited from the `Plot` class, are listed in table 29.3.

The following default values are used where necessary:

Name:	Value:
DEFAULT_INTERIOR_GAP	0.25 (25 percent)
DEFAULT_START_ANGLE	90.0
DEFAULT_LABEL_FONT	new Font("SansSerif", Font.PLAIN, 10);
DEFAULT_LABEL_PAINT	Color.black;
DEFAULT_LABEL_BACKGROUND_PAINT	new Color(255, 255, 192);
DEFAULT_LABEL_GAP	0.10 (10 percent)

Attribute:	Description:
interiorGap	The amount of space to leave blank around the outside of the pie, expressed as a percentage of the chart height and width. Extra space is added for the labels.
circular	A flag that controls whether the pie chart is constrained to be circular, or allowed to take on an elliptical shape to fit the available space.
startAngle	The angle of the first pie section, expressed in degrees (0 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 (`Rotation.CLOCKWISE`) or anticlockwise (`Rotation.ANTI_CLOCKWISE`) direction.
sectionPaint	The paint used for all sections (usually `null`).
sectionPaintList	The paint used for each section, unless overridden by *sectionPaint*.
baseSectionPaint	The default paint, used when no other setting is specified.
sectionOutlinePaint	The outline paint used for all sections (usually `null`).
sectionOutlinePaintList	The outline paint used for each section.
baseSectionOutlinePaint	The default outline paint, used when no other setting is specified.
sectionOutlineStroke	The outline stroke used for all sections (usually `null`).
sectionOutlineStrokeList	The outline stroke used for each section.
baseSectionOutlineStroke	The default outline stroke, used when no other setting is specified.
shadowPaint	The shadow paint.
shadowXOffset	The x-offset for the shadow effect.
shadowYOffset	The y-offset for the shadow effect.
explodePercentages	The amount (percentage) to "explode" each pie section.
labelGenerator	The section label generator, an instance of `PieSectionLabelGenerator`.
labelFont	The font for the section labels.
labelPaint	The color for the section labels.
labelBackgroundPaint	The background color for the section labels.
maximumLabelWidth	The maximum label width as a percentage of the plot width.
labelGap	The gap for the section labels.
labelLinkMargin	The label link margin.
labelLinkPaint	The `Paint` used for the lines that connect the pie sections with their corresponding labels.
labelLinkStroke	The `Stroke` used for the lines that connect the pie sections to their corresponding labels.
toolTipGenerator	A plug-in tool tip generator.
urlGenerator	A plug-in URL generator (for image map generation).
pieIndex	The index for this plot (only used by the `MultiplePiePlot` class).

Table 29.3: Attributes for the `PiePlot` class

29.19.4 Methods

To replace the dataset being used by the plot:

> `public void setDataset(PieDataset dataset);`
> Replaces the dataset being used by the plot (this triggers a `DatasetChangeEvent`).

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.

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). This is the same encoding used by Java's `Arc2D` class.

To control the direction (clockwise or anticlockwise) of the sections in the pie chart:

> `public void setDirection(Rotation direction);`
> Sets the direction of the sections in the pie chart. Use one of the constants `Rotation.CLOCKWISE` (the default) and `Rotation.ANTICLOCKWISE`.

To control the amount of space around the pie chart:

> `public void setInteriorGapPercent(double percent);`
> Sets the amount of space inside the plot area.

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).

If you are displaying your pie chart in a `ChartPanel` and you want to customise the tooltip text, you can register your own tool tip generator with the plot:

> `public void setToolTipGenerator(PieToolTipGenerator generator);`
> Registers a tool tip generator with the pie plot. You can set this to `null` if you do not require tooltips.

29.19.5 Section Colors

The colors used to fill the sections in a pie chart are fully customisable. To set the color used to fill a particular section:

> `public void setSectionPaint(int section, Paint paint);`
> Sets the paint used to fill a particular section in the chart and sends a `PlotChangeEvent` to all registered listeners.

In a similar way, you can control the paint and stroke used to outline individual sections in the chart. To set the outline paint:

> `public void setSectionOutlinePaint(int section, Paint paint);`
> Sets the paint used to outline a particular section in the chart and sends a `PlotChangeEvent` to all registered listeners.

To set the outline stroke:

> `public void setSectionOutlineStroke(int section, Stroke stroke);`
> Sets the stroke used to outline a particular section in the chart and sends a `PlotChangeEvent` to all registered listeners.

29.19.6 Shadow Effect

The pie plot will draw a “shadow” effect. To set the paint used to draw the shadow:

> `public void setShadowPaint(Paint paint);`
> Sets the paint used to draw the “shadow” effect. If you set this to `null`, no shadow effect will be drawn.

To set the x-offset for the shadow effect:

> `public void setShadowXOffset(double offset);`
> Sets the x-offset (in Java2D units) for the shadow effect.

To set the y-offset for the shadow effect:

> `public void setShadowYOffset(double offset);`
> Sets the y-offset (in Java2D units) for the shadow effect.

29.19.7 Exploded Sections

It is possible to “explode” sections of the pie chart. The `PieChartDemo2` application (included in the JFreeChart distribution) provides a demo.

29.19.8 Section Labels

Section labels are generated by a plugin class that implements the `PieItemLabelGenerator` interface.

To set a new generator:

> `public void setLabelGenerator(PieSectionLabelGenerator generator);`
> Sets the label generator for the plot and sends a `PlotChangeEvent` to all registered listeners.

To set the color of the lines connecting the pie sections to their corresponding labels:

> `public void setLabelLinkPaint(Paint paint);`
> Sets the `Paint` used for the lines connecting the pie sections to their corresponding labels and sends a `PlotChangeEvent` to all registered listeners.

To set the line style for the linking lines:

> `public void setLabelLinkStroke(Stroke stroke);`
> Sets the `Stroke` used for the lines connecting the pie sections to their corresponding labels and sends a `PlotChangeEvent` to all registered listeners.

At the current time, there is no facility to hide the linking lines.

29.19.9 Notes

Some points to note:

- 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 `org.jfree.chart.demo` package provides a simple pie chart demonstration.

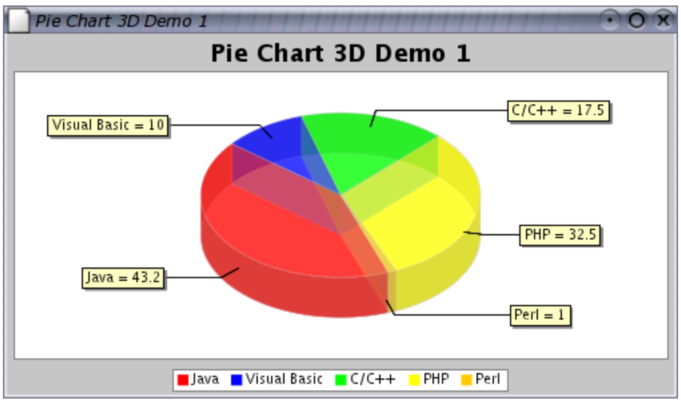
See Also

`PieDataset`, `PieItemLabelGenerator`, `Plot`.

29.20 PiePlot3D

29.20.1 Overview

An extension of the `PiePlot` class that draws pie charts with a 3D effect.



29.20.2 Notes

This class does not yet support the "exploded" sections that can be displayed by the regular pie charts.

29.21 Plot

29.21.1 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 29.6 illustrates the plot class hierarchy.

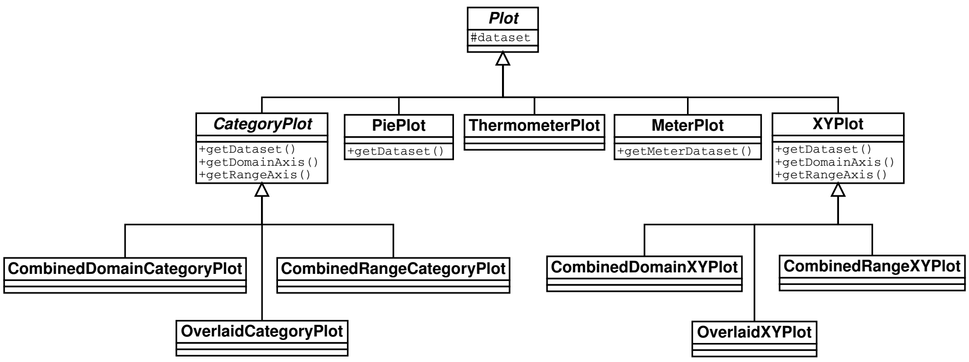


Figure 29.6: Plot classes

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.

29.21.2 Constructors

This class is abstract, so the constructors are `protected`. You cannot create an instance of this class directly, you must use a subclass.

29.21.3 Attributes

This class maintains the following attributes:

Attribute:	Description:
insets	The amount of space to leave around the outside of the plot.
outlineStroke	The `Stroke` used to draw an outline around the plot area.
outlinePaint	The `Paint` used to draw an outline around the plot area.
backgroundPaint	The `Paint` used to draw the background of the plot area.
backgroundImage	An image that is displayed in the background of the plot (optional).
backgroundImageAlignment	The image alignment.
backgroundAlpha	The alpha transparency value used when coloring the plot's background, and also when drawing the background image (if there is one).
foregroundAlpha	The alpha transparency used to draw items in the plot's foreground.
noDataMessage	A string that is displayed by some plots when there is no data to display.
noDataMessageFont	The `Font` used to display the "no data" message.
noDataMessagePaint	The `Paint` used to display the "no data" message.
drawingSupplier	The drawing supplier.
dataAreaRatio	The aspect ratio for the data area.
datasetGroup	The dataset group (to be used for synchronising dataset access).

Table 29.4: Attributes for the `Plot` class

All subclasses will inherit these core attributes.

29.21.4 Usage

To customise the appearance of a plot, you first obtain a reference to the plot as follows:

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

With this reference, you can change the appearance of the plot by modifying it's attributes. For example:

```
plot.setBackgroundPaint(Color.lightGray); 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.

29.21.5 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 = chart.getPlot(); plot.setBackgroundPaint(Color.lightGray);
```

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).

29.21.6 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`.

29.21.7 Notes

Refer to specific subclasses for information about setting the colors, shapes and line styles for data drawn by the plot.

29.22 PlotOrientation

29.22.1 Overview

Used to represent the orientation of a plot (in particular, the CategoryPlot and XYPlot classes). There are two values, as listed in table 29.5.

Class:	Description:
PlotOrientation.HORIZONTAL	A "horizontal" orientation.
PlotOrientation.VERTICAL	A "vertical" orientation.

Table 29.5: Plot orientation values

The orientation corresponds to the "direction" of the range axis. So, for example, a bar chart with a vertical orientation will display vertical bars, while a bar chart with a horizontal orientation will display horizontal bars.

29.22.2 Notes

For interesting effects, in addition to changing the orientation of a chart you can:

- change the location of the chart's axes;
- invert the scale of the axes.

29.23 PlotRenderingInfo

29.23.1 Overview

This class is used to record information about the individual elements in a single rendering of a plot. See also the ChartRenderingInfo class.

29.24 PolarPlot

29.24.1 Overview

A plot that is used to display data from an XYDataset using polar coordinates—see figure 29.7 for an example.

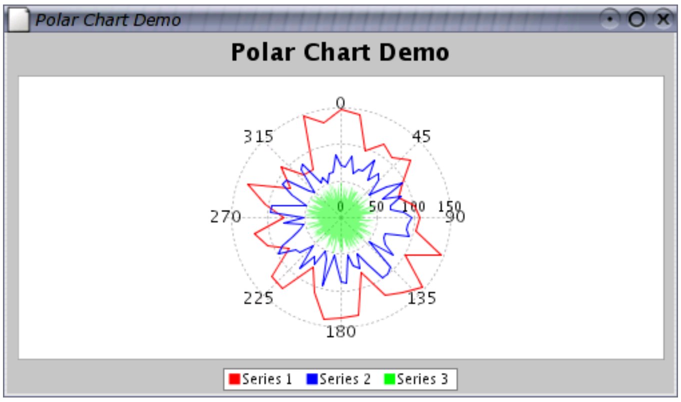


Figure 29.7: A polar chart

The items in the plot are drawn by a PolarItemRenderer.

29.24.2 Usage

There is a demo application (`PolarChartDemo.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) that illustrates the use of this class.

29.24.3 Notes

Some points to note:

- instances of this class are cloneable and serializable.

29.25 ThermometerPlot

29.25.1 Overview

A plot that displays a single value in a thermometer-style representation.

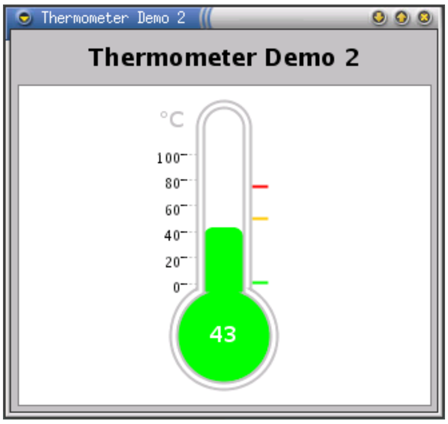


Figure 29.8: A thermometer chart

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.

29.25.2 Constructors

To create a new `ThermometerPlot`:

> `public ThermometerPlot(ValueDataset dataset);`
> Creates a thermometer with default settings, using the supplied dataset.

29.25.3 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.

29.25.4 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.

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 `org.jfree.chart.demo` package provides a working example of this class.

29.26 ValueMarker

29.26.1 Overview

A *value marker* is used to indicate a constant value against the domain or range axis for a `CategoryPlot` or an `XYPlot`. This class extends the `Marker` class.

29.26.2 Usage

There is a demo application (`MarkerDemo1.java`) included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory) that illustrates the use of this class.

29.26.3 Notes

Some points to note:

- this class is `Cloneable` and `Serializable`.

29.27 XYPlot

29.27.1 Overview

Draws a visual representation of data from an `XYDataset`, where the domain axis measures the x-values and the range axis measures the y-values.

The type of plot is typically displayed using a vertical orientation, but it is possible to change to a horizontal orientation which can be useful for certain applications.

29.27.2 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:

Figure 29.9: The plot regions

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).

29.27.3 Renderers

The `XYPlot` class delegates drawing of individual data items to an `XYItemRenderer`. A number of renderer implementations are available (and you are free to develop your own, of course):

- `CandlestickRenderer`;
- `ClusteredXYBarRenderer`;
- `HighLowRenderer`;
- `StandardXYItemRenderer`;
- `XYAreaRenderer`;
- `XYBarRenderer`;
- `XYBubbleRenderer`;
- `XYDifferenceRenderer`;

29.27.4 Axis Offsets

It is possible to specify the amount by which the plot's axes are offset from the data area. By default, there is no offset, but you can change this easily, for example:

```
plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
```

29.27.5 Location of Axes

The plot's axes can appear at the top, bottom, left or right of the plot area. The location for an axis is specified using the `AxisLocation` class, which combines two possible locations within each option—which one is actually used depends on the orientation (horizontal or vertical) of the plot.

For "vertical" plots (the usual default), the domain axis will appear at the top or bottom of the plot area, and the range axis will appear at the left or right of the plot area. For "horizontal" plots, the domain axis will appear at the left or right of the plot area, and the range axis will appear at the top or bottom of the plot area.

To set the location for the domain axis:

> `public void setDomainAxisLocation(AxisLocation location);`
> Sets the location for the domain axis and sends a `PlotChangeEvent` to all registered listeners.

Similarly, to set the location for the range axis:

> `public void setRangeAxisLocation(AxisLocation location);`
> Sets the range axis location and sends a `PlotChangeEvent` to all registered listeners.

For example, to display the range axis on the right side of a chart:

```
plot.setRangeAxisLocation(AxisLocation.BOTTOM_OR_RIGHT);
```

This assumes the plot orientation is vertical, if it changes to horizontal the axis will be displayed at the bottom of the chart.

29.27.6 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 gridlines. 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.

29.27.7 Markers

Markers are used to highlight particular values along the domain axis or the range axis for a plot. Typically, a marker will be represented by a solid line perpendicular to the axis against which it is measured, although custom renderers can alter this default behaviour.

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 (assuming a vertical orientation for 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 (assuming a vertical orientation for 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.

29.27.8 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 = (XYPlot) chart.getPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

To clear all annotations:

```
plot.clearAnnotations();
```

29.27.9 Constructors

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.

29.27.10 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.

29.27.11 Notes

It is possible to display time series data with `XYPlot` by employing a `DateAxis` in place of the usual `NumberAxis`. In this case, the x-values are interpreted as "milliseconds since 1-Jan-1970" as used in `java.util.Date`.

See Also

`Plot`, `XYItemRenderer`, `CombinedDomainXYPlot`, `CombinedRangeXYPlot`.

Chapter 30

Package: org.jfree.chart.renderer

30.1 Overview

This package contains interfaces and classes that are used to implement renderers, plug-in objects that are responsible for drawing individual data items on behalf of a `CategoryPlot` or an `XYPlot`.

Renderers offer a lot of scope for changing the appearance of your charts, either by changing the attributes of an existing renderer, or by implementing a completely new renderer.

30.2 AbstractCategoryItemRenderer

30.2.1 Overview

A base class that can be used to implement a new `CategoryItemRenderer`.

30.2.2 Constructors

The default constructor creates a renderer with no tooltip generator and no URL generator. The constructor is `protected`.

30.2.3 Attributes

The attributes maintained by this class are listed in Table 30.1.

30.2.4 Methods

The following method is called once every time the chart is drawn:

```
public CategoryItemRendererState initialise(Graphics2D g2, Rectangle2D
dataArea, CategoryPlot plot, Integer index, PlotRenderingInfo info);
```

Attribute:	Description:
plot	The CategoryPlot that the renderer is assigned to.
toolTipGenerator	The CategoryItemLabelGenerator that generates tool tips for ALL series (can be `null`).
toolTipGeneratorList	A list of CategoryItemLabelGenerator objects used to create tool tips for individual series.
baseToolTipGenerator	The base CategoryItemLabelGenerator used to create tool tips when there is no other generator available.
labelGenerator	The CategoryItemLabelGenerator that generates item labels for ALL series (can be `null`).
labelGeneratorList	A list of CategoryItemLabelGenerator objects used to create item labels for individual series. If `null`, the *baseLabelGenerator* is used instead.
baseLabelGenerator	The base CategoryItemLabelGenerator used to create item labels when no other generator is available.
itemURLGenerator	The CategoryURLGenerator that applies to ALL series.
itemURLGeneratorList	A list of CategoryURLGenerator objects that apply to individual series. If `null`, the *baseItemURLGenerator* is used instead.
baseItemURLGenerator	The base CategoryURLGenerator, used when no other generator is available.

Table 30.1: Attributes for the `AbstractCategoryItemRenderer` *class*

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 CategoryDataset) is cached by the renderer in the `initialise(...)` method.

To get the renderer type:

```
public RangeType getRangeType();
```

Returns the range type for the renderer (`STANDARD` or `STACKED`).

To draw the plot background:

```
public void drawBackground(Graphics2D g2, CategoryPlot plot, Rectangle2D
dataArea);
```

Draws the plot background. Some renderers will choose to override this method, but for most the default behaviour is OK.

To draw the plot outline:

```
public void drawOutline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the plot outline. Some renderers will choose to override this method, but for most the default behaviour is OK.

To draw a domain gridline:

```
public void drawDomainGridline(Graphics2D g2, CategoryPlot plot, Rectangle2D
dataArea, double value);
```

Draws a domain gridline at the specified value.

To draw a range gridline:

`public void drawRangeGridline(Graphics2D g2, CategoryPlot plot, ValueAxis axis, Rectangle2D dataArea, double value);`
Draws a range gridline at the specified value.

To draw a range marker:

`public void drawRangeMarker(Graphics2D g2, CategoryPlot plot, ValueAxis axis, Marker marker, Rectangle2D dataArea);`
Draws a range marker.

To get a legend item:

`public LegendItem getLegendItem(int datasetIndex, int series);`
Returns a legend item for the specified series. The `datasetIndex` is zero for the primary dataset, and 1..N for the secondary datasets.

To get the `CategoryItemLabelGenerator` for a data item:

`public CategoryItemLabelGenerator getItemLabelGenerator(int row, int column);`
Returns the item label generator for a specific data item. By default, this method just calls the `getSeriesItemLabelGenerator()` method.

To get the `CategoryItemLabelGenerator` for a series:

`public CategoryItemLabelGenerator getSeriesItemLabelGenerator(int series);`
Returns the item label generator for a series. This method returns the *itemLabelGenerator* if it is set, otherwise it looks up the *itemLabelGeneratorList* to get a generator specific to the series. If the series-specific generator is `null`, the *baseItemLabelGenerator* is returned.

To get the `CategoryURLGenerator` for a data item:

`public CategoryURLGenerator getItemURLGenerator(int row,int column);`
Returns the item URL generator for a specific data item. By default, this method just calls the `getSeriesItemURLGenerator(...)` method.

To get the `CategoryURLGenerator` for a series:

`public CategoryURLGenerator getSeriesItemURLGenerator(int series);`
Returns the item URL generator for a series. This method returns the *itemURLGenerator* if it is set, otherwise it looks up the *itemURLGeneratorList* to get a generator specific to the series. If the series-specific generator is `null`, the *baseItemURLGenerator* is returned.

To get the row count:

`public int getRowCount();`
Returns the row count.

To get the column count:

`public int getColumnCount();`
Returns the column count.

30.2.5 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

30.3 AbstractRenderer

30.3.1 Overview

An abstract class that provides common services for renderer implementations.

This base class is extended by both the `AbstractCategoryItemRenderer` class and the `AbstractXYItemRenderer` class.

30.3.2 Attributes

The attributes maintained by the `AbstractRenderer` class are listed in Table 30.2.

Attribute:	Description:
paint	The paint that applies to ALL series (`null` permitted).
paintList	A list of paints that apply to individual series (only referenced if *paint* is `null`).
basePaint	The paint that is used if there is no other setting.
outlinePaint	The outline paint that applies to ALL series (`null` permitted).
outlinePaintList	A list of outline paints that apply to individual series (only referenced if *outlinePaint* is `null`).
baseOutlinePaint	The outline paint that is used if there is no other setting.
stroke	The stroke that applies to ALL series (`null` permitted).
strokeList	A list of stroke objects that apply to individual series (only referenced if *stroke* is `null`).
baseStroke	The stroke that is used if there is no other setting.
outlineStroke	The outline stroke that applies to ALL series (`null` permitted).
outlineStrokeList	A list of outline strokes that apply to individual series (only referenced if *outlineStroke* is `null`).
baseOutlineStroke	The outline stroke that is used if there is no other setting.
shape	The shape that applies to ALL series (`null` permitted).
shapeList	A list of shapes that apply to individual series (only referenced if *shape* is `null`).
baseShape	The shape that is used if there is no other setting.

Table 30.2: Attributes for the `AbstractRenderer` class

30.3.3 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 `PaintList`).

There is a default mechanism to automatically populate the paint list with default colors (using the `DrawingSupplier` interface). However, you can manually

Attribute:	**Description:**
itemLabelsVisible	The flag that applies to ALL series (`null` permitted).
itemLabelsVisibleList	A list of flags that apply to individual series (only referenced if *itemLabelsVisible* is `null`).
baseItemLabelsVisible	The flag that is used if there is no other setting.
itemLabelFont	The font that applies to ALL series (`null` permitted).
itemLabelFontList	A list of fonts that apply to individual series (only referenced if *itemLabelFont* is `null`).
baseItemLabelFont	The font that is used if there is no other setting.
itemLabelPaint	The paint that applies to ALL series (`null` permitted).
itemLabelPaintList	A list of paints that apply to individual series (only referenced if *itemLabelPaint* is `null`).
baseItemLabelPaint	The font that is used if there is no other setting.
itemLabelAnchor	The anchor that applies to ALL series (`null` permitted).
itemLabelAnchorList	A list of anchors that apply to individual series (only referenced if *itemLabelAnchor* is `null`).
baseItemLabelAnchor	The anchor that is used if there is no other setting.
itemLabelTextAnchor	The text anchor that applies to ALL series (`null` permitted).
itemLabelTextAnchorList	A list of text anchors that apply to individual series (only referenced if *itemLabelTextAnchor* is `null`).
baseItemLabelTextAnchor	The text anchor that is used if there is no other setting.
itemLabelRotationAnchor	The rotation anchor that applies to ALL series (`null` permitted).
itemLabelRotationAnchorList	A list of rotation anchors that apply to individual series (only referenced if *itemLabelRotationAnchor* is `null`).
baseItemLabelRotationAnchor	The anchor that is used if there is no other setting.
itemLabelAngle	The angle that applies to ALL series (`null` permitted).
itemLabelAngleList	A list of angles that apply to individual series (only referenced if *itemLabelAnchor* is `null`).
baseItemLabelAngle	The angle that is used if there is no other setting.

Table 30.3: Attributes for the `AbstractRenderer` class

update the paint list 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();
```

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);
```

30.3.4 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);
}
```

30.3.5 Equals, Cloning and Serialization

This class overrides the equals(...) method. *TO DO: review equality tests for `Paint` and `Stroke` objects.*

30.4 AbstractXYItemRenderer

30.4.1 Overview

A convenient base class for creating new `XYItemRenderer` implementations.

30.4.2 Constructors

This class provides a default constructor which allocates storage for the label generator(s), the tool tip generator(s) and the URL generator.

> `protected AbstractXYItemRenderer();`
> Creates a new renderer.

30.4.3 Initialisation

Each time a chart is drawn, the plot will initialise the renderer by calling the following method:

> `public XYItemRendererState initialise(...)`
> Initialises the renderer and returns a state object that the plot will pass to all subsequent calls to the `drawItem()` method. The state object is discarded once the chart is fully drawn.

30.4.4 The Pass Count

The *pass count* refers to the number of times the `XYPlot` scans through the dataset passing individual data items to the renderer for drawing. Most renderers require only a single pass through the dataset, but some will use a second pass to overlay shapes (for example) over previously drawn items.

The plot will call the following method to determine how many passes the renderer requires:

> `public int getPassCount();`
> Returns `1` to indicate that the renderer requires only a single pass through the dataset.

Renderers that require more than one pass through the dataset should override this method.

30.4.5 Domain and Range Markers

A default method is supplied for displaying a *domain marker* as a line on the plot:

> `public void drawDomainMarker(...);`
> Draws a line perpendicular to the domain axis to represent a `Marker`.

A default method is supplied for displaying a *range marker* as a line on the plot:

> `public void drawRangeMarker(...);`
> Draws a line perpendicular to the range axis to represent a `Marker`.

Most renderers will use these methods by default, but some may override them.

30.4.6 Grid Bands

It is possible to fill the space between alternate grid lines with a different color to create a "band" effect.

30.4.7 Methods

To find out the range type for the renderer:

> `public RangeType getRangeType();`
> Returns the range type for the renderer, which affects the auto-range calculation for the axis that the renderer is mapped to.

To create a legend item for a series (this method is called by the plot):

> `public LegendItem getLegendItem(int index, int series);`
> Returns a legend item that represents the specified series. The `index` argument tells the renderer which dataset it is rendering (only the plot tracks this)—`0` for the primary dataset, or `n+1` for a secondary dataset (where `n` is the index of the secondary dataset).

30.4.8 Notes

Some points to note:

- this class provides a property change mechanism to support the requirements of the `XYItemRenderer` interface;

See Also
`XYItemRenderer`, `XYPlot`.

30.5 AreaRenderer

30.5.1 Overview

A *category item renderer* that represents each item in a `CategoryDataset` using a polygon that fills the area between the x-axis and the data point.
This renderer is designed for use with the `CategoryPlot` class.

30.5.2 Notes

Some notes:

- the `createAreaChart(...)` method in the `ChartFactory` class will create a default chart that uses this renderer.
- this class extends `AbstractCategoryItemRenderer`.

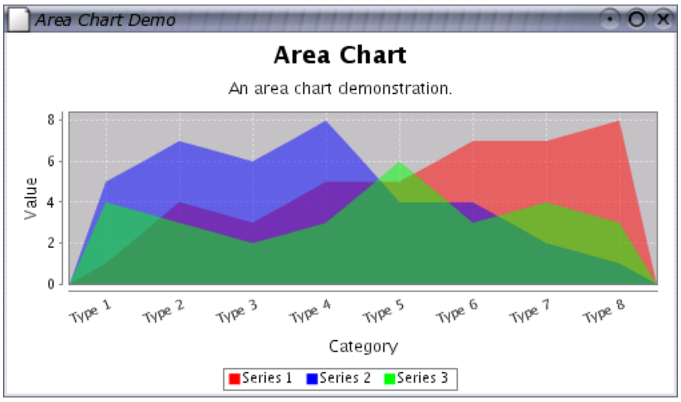


Figure 30.1: An area chart

See Also

XYAreaRenderer.

30.6 BarRenderer

30.6.1 Overview

A *bar renderer* is used with a CategoryPlot to create bar charts from data in a CategoryDataset. Figure 30.2 shows an example of a bar chart with a vertical orientation and figure 30.3 shows an example of a bar chart with a horizontal orientation.

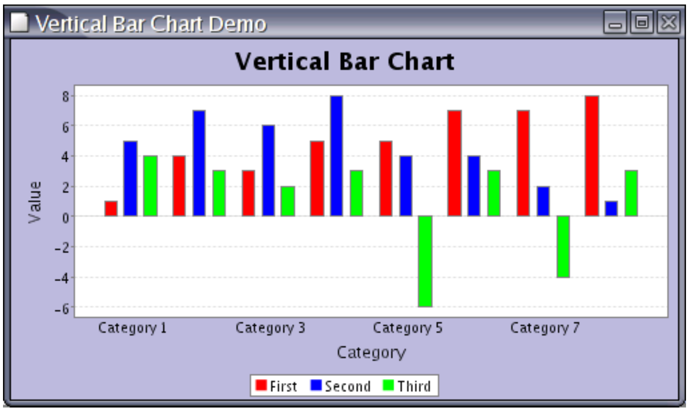


Figure 30.2: A vertical bar chart

This class extends AbstractCategoryItemRenderer.

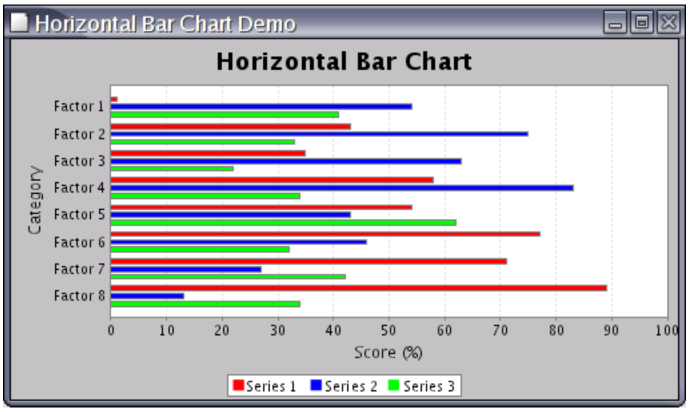


Figure 30.3: A horizontal bar chart

30.6.2 The Bar Width

The renderer automatically calculates the width of the bars to fit the available space for the plot, so you cannot directly control how wide the bars are. However, the bar width is a function of the following attributes that you can control:

- the *lowerMargin*, *upperMargin* and *categoryMargin* attributes, all defined by the `CategoryAxis`;
- the *item margin* attribute belonging to the renderer (see below).

The *item margin* attribute controls 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.

30.6.3 Appearance

The dynamic bar width calculation can result in very wide bars if you have only a few data values in a chart. If you would like to specify a "cap" for the bar width, use this method:

> `public void setMaxBarWidth(double percent);`
> Sets the maximum bar width as a percentage of the axis length. For example, setting this to `0.05` will ensure that the bars never exceed five percent of the length of the axis.

You can specify whether or not bars are drawn with an outline using:

> `public void setDrawBarOutline(boolean draw);`
> Sets a flag that controls whether or not an outline is drawn around each

bar. The paint and stroke used for the bar outline is specified using methods in the superclass.

To provide better support for the use of `GradientPaint` objects to color the bars drawn by this renderer, you can specify a *transformer* that will dynamically adjust the `GradientPaint` to fit each bar, using the following method:

> `public void setGradientPaintTransformer(GradientPaintTransformer transformer);`
> Sets the transformer. If this is set to `null`, any `GradientPaint` objects will be used in their raw form (i.e. with fixed coordinates).

For very small data values (relative to the axis range), you can have bars with a length of less than 1 pixel (on-screen)—when the value gets too small, the bar will disappear. If you want to ensure that a line is always drawn so that the small bar is visible, you can specify a minimum bar length with this method:

> `public void setMinimumBarLength(double min);`
> Sets the minimum length that will be used for a bar, specified in Java 2D units. You can set this to 1.0, for example, to ensure that very short bars do not disappear.

30.6.4 Item Labels

This renderer supports the display of item labels. Due to the rectangular nature of the bars, the renderer calculates anchor points are a arranged as shown in figure 30.4.

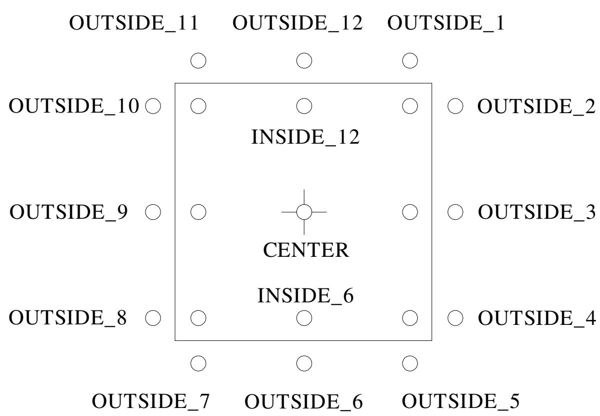


Figure 30.4: Item Label Anchors for Bars

To control the amount of space between item labels (if they are visible) and the edge of the bar:

```
public void setItemLabelAnchorOffset(double offset);
```

Sets the offset (in Java2D units) between the edge of the bar and the item label anchor point.

30.6.5 Methods

This class implements all the methods in the `CategoryItemRenderer` interface.

30.6.6 Notes

Some points to note:

- the `ChartFactory` class uses this renderer when it constructs bar charts.
- the `BarChartDemo.java` class, included in the JFreeChart distribution, is one example that uses this renderer.

See Also

`StackedBarRenderer`, `BarRenderer3D`, `StackedBarRenderer3D`.

30.7 BarRenderer3D

30.7.1 Overview

A renderer that draws items from a `CategoryDataset` using bars with a 3D effect. Figure 30.5 shows the renderer being used with a plot that has a vertical orientation and figure 30.6 shows the renderer being used with a plot that has a horizontal orientation.

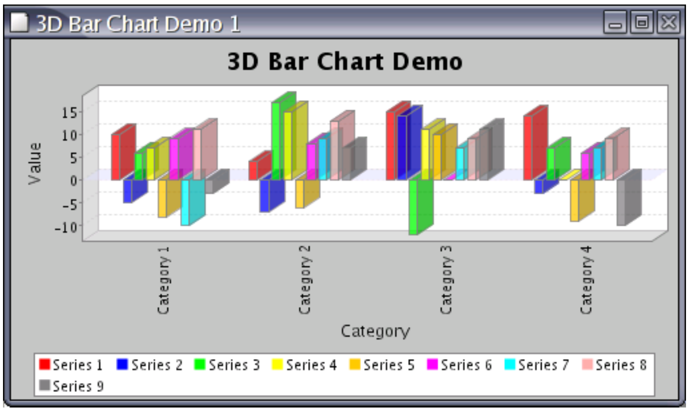


Figure 30.5: An example of the BarRenderer3D class at work

This renderer is designed for use with the `CategoryPlot` class.

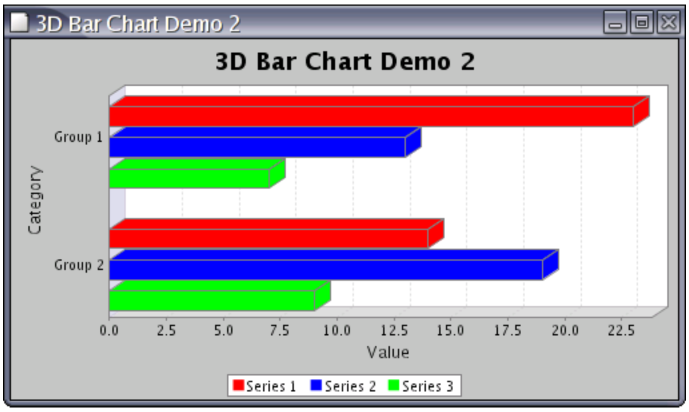


Figure 30.6: Another 3D bar chart

30.7.2 Notes

Some points to note:

- this class implements the `CategoryItemRenderer` interface.
- the `BarChart3DDemo1` and `BarChart3DDemo2` applications (included in the JFreeChart distribution) provide demonstrations of this renderer in use.

30.8 BoxAndWhiskerRenderer

30.8.1 Overview

A renderer that is used to create a *box-and-whisker plot* within the `CategoryPlot` framework.

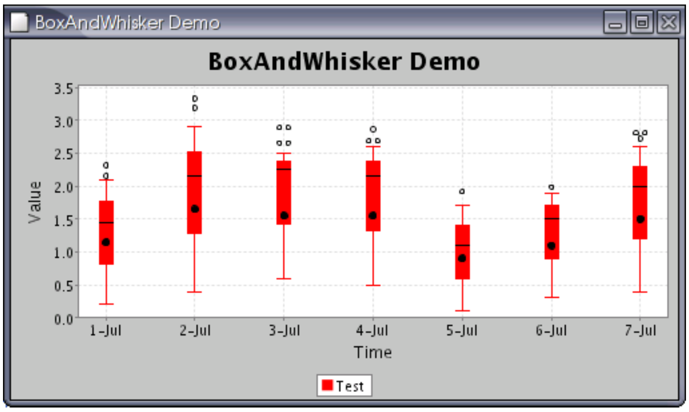


Figure 30.7: A box-and-whisker plot

30.8.2 Notes

The `BoxAndWhiskerDemo` application, included in the JFreeChart distribution, provides an example of this type of plot.

30.9 CandlestickRenderer

30.9.1 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).

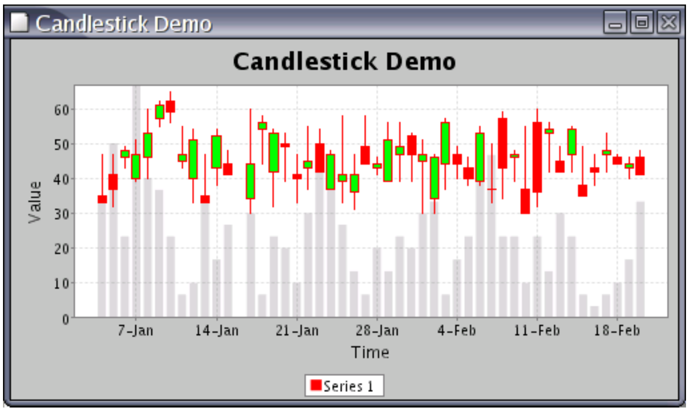


Figure 30.8: A sample chart using `CandlestickRenderer`

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.

30.9.2 Constructors

To create a new renderer:

```
public CandlestickRenderer(double candleWidth);
```
Creates a new renderer.

30.9.3 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.

30.9.4 Notes

This renderer requires a `HighLowDataset`.

30.10 CategoryItemRenderer

30.10.1 Overview

A *category item renderer* is an object that is assigned to a `CategoryPlot` and assumes responsibility for drawing the visual representation of individual data items in a dataset. This interface defines the methods that must be provided by all category item renderers—the plot will only use the methods defined in this interface.

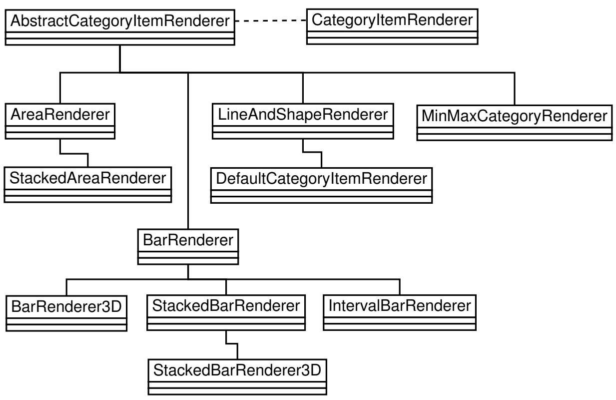


Figure 30.9: Category item renderers

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:
AreaRenderer	Used to create area charts.
BarRenderer	Represents data using bars (anchored at zero).
BarRenderer3D	Represents data using bars (anchored at zero) with a 3D effect.
StackedBarRenderer	Used to create a stacked bar charts.
IntervalBarRenderer	Draws intervals using bars. This renderer can be used to create simple Gantt charts.
LineAndShapeRenderer	Draws lines and/or shapes to represent data.

30.10.2 Methods

The interface defines an initialisation method:

> `public CategoryItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, CategoryPlot plot, Integer index, PlotRenderingInfo info);`
> This method is called exactly once at the start of every chart redraw. The method returns a state object that the plot will pass to the `drawItem()` method for each data item that the renderer needs to draw. Thus, 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 RangeType getRangeType();`
> Returns the range type (`STANDARD` or `STACKED`) for the renderer.

The most important method is the one that actually draws a data item:

> `public void drawItem(...);`
> Draws one item on a category plot. The `CategoryPlot` class will iterate through the data items, passing them to the renderer one at a time.

30.10.3 Item Labels

An *item label* is a short text string that can be displayed near each data item in a chart. Whenever the renderer requires an item label, it obtains a label generator via the following method:

> `public CategoryItemLabelGenerator getLabelGenerator(int series, int item);`
> Returns the label generator for the specified data item. In theory, this method could return a different generator for each item but, in practice, it will often return the same generator for every item (or one generator per series). The method can return `null` if no generator has been set for the renderer—in this case, no item labels will be displayed.

To set a generator that will be used for all data items in the chart:

`public void setLabelGenerator(CategoryItemLabelGenerator generator);`
Sets the label generator that will be used for ALL data items in the chart, and sends a `RendererChangeEvent` to all registered listeners. Set this to `null` if you prefer to set the generator on a "per series" basis.

To set a generator for a particular series:

`public void setSeriesLabelGenerator(int series, CategoryItemLabelGenerator generator);`
Sets the item label generator for the specified series. If `null`, the *baseItemLabelGenerator* will be used.

To make item labels visible for ALL series:

`public void setItemLabelsVisible(boolean visible);`
Sets the flag that controls whether or not item labels are visible for all series drawn by this renderer. If you prefer to set the visibility on a *per series* basis, you need to set this flag to `null` (see the next method).

`public void setItemLabelsVisible(Boolean visible);`
Sets the flag that controls whether or not item labels are visible for all series drawn by this renderer. Set this to *null* if you prefer to set the visibility on a *per series* basis.

To control the visibility of item labels for a particular series:

`public void setSeriesItemLabelsVisible(int series, boolean visible);`
Sets a flag that controls whether or not item labels are visible for the specified series.

`public void setSeriesItemLabelsVisible(int series, Boolean visible);`
Sets a flag that controls whether or not item labels are visible for the specified series. If this is set to *null*, the *baseItemLabelsVisible* flag determines the visibility.

The position of the item labels is set using the following methods (one applies to positive data items and the other applies to negative data items):

`public void setPositiveItemLabelPosition(ItemLabelPosition position);`
Sets the position for labels for data items where the y-value is positive.

`public void setNegativeItemLabelPosition(ItemLabelPosition position);`
Sets the position for labels for data items where the y-value is negative.

30.10.4 Tooltips

A *tool tip* is a short text string that is displayed temporarily in a GUI while the mouse pointer hovers over a particular item in a chart. Whenever the renderer requires a text string for a tool tip, it calls the following method:

`public CategoryToolTipGenerator getToolTipGenerator(int series, int item);`
Returns the tool tip generator for the specified data item (possibly `null`).

You can register a generator with the renderer using:

`public void setToolTipGenerator(CategoryToolTipGenerator generator);`
Sets the tool tip generator that will be used for ALL data items in the chart, and sends a `RendererChangeEvent` to all registered listeners.

30.10.5 URL Generation

The ChartEntity objects created by the renderer for each data item can have a URL associated with them. To provide flexibility, URLs are generated using a mechanism that is very similar to the tooltips mechanism.

> *URLs are only used in HTML image maps at present. If you are not generating HTML image maps, then you should leave the URL generators set to `null`.*

You can associate a CategoryURLGenerator with the renderer using this method:

> `public void setItemURLGenerator(CategoryURLGenerator generator);`
> Sets the generator that will be used to generate URLs for items in ALL series.

It is possible to specify a different URL generator for each series by first setting the generator in the previous method to `null` then using the following method to assign a generator to each series independently:

> `public void setSeriesItemURLGenerator(int series, CategoryURLGenerator generator);`
> Sets the generator for the items in a particular series.

In most cases, a single generator for all series will suffice.

30.10.6 Notes

Some points to note:

- 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`.

30.11 CategoryItemRendererState

30.11.1 Overview

This class records state information for a `CategoryItemRenderer` during the process of drawing a chart.

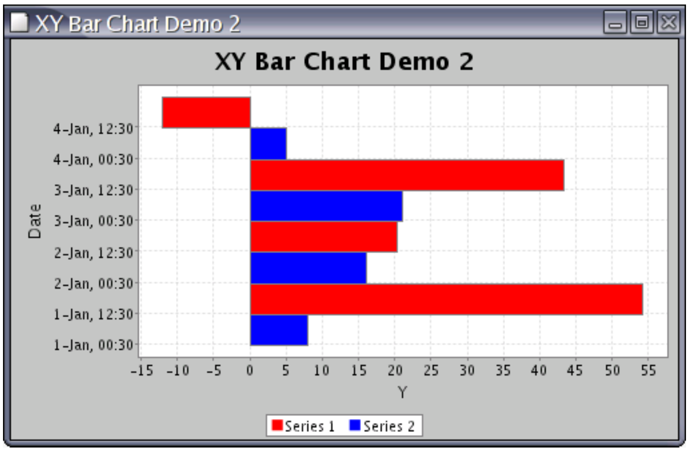
Recall that the plot uses a renderer to draw the individual data items in a chart. In the plot's `render()` method, a call is made to the renderer's `initialise()` method, which returns a state object. Subsequently, for every call the plot makes to the renderer's `drawItem()` method, it passes in the same state object (which can be updated with new state information during the rendering).

This scheme is designed to allow two or more different threads to use a single renderer to draw a chart to different output targets simultaneously.

30.12 ClusteredXYBarRenderer

30.12.1 Overview

An *XY bar renderer* draws items from an IntervalXYDataset in the form of bars.



This renderer is designed to work with an XYPlot.

30.12.2 Constructors

The only constructor takes no arguments.

30.12.3 Methods

The `drawItem()` method handles the rendering of a single item for the plot.

30.12.4 Notes

This renderer casts the dataset to IntervalXYDataset, so you should ensure that the plot is supplied with the correct type of data. It would probably be a good idea to merge this class with the XYBarRenderer class, but this hasn't been done yet.

30.13 DefaultPolarItemRenderer

30.13.1 Overview

A default renderer for use by the PolarPlot class (implements the PolarItemRenderer interface).

30.14 GanttRenderer

30.14.1 Overview

A renderer that is used to draw simple Gantt charts—an example is shown in figure 30.10.

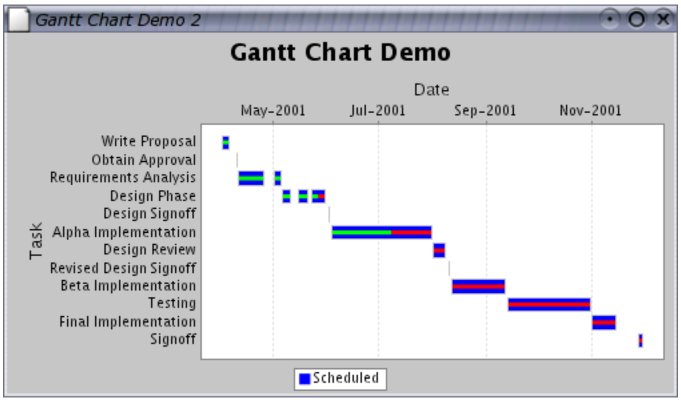


Figure 30.10: A Gantt chart

The renderer is used with the `CategoryPlot` class and accesses data via the `GanttCategoryDataset` interface.

30.14.2 Methods

The renderer can highlight the "percentage complete" for a task, provided that this information is specified in the dataset. The colors used for this indicator are set with the following methods:

> `public void setCompletePaint(Paint paint);`
> Sets the `Paint` used to draw the portion of the task that is completed and sends a `RendererChangeEvent` to all registered listeners.

> `public void setIncompletePaint(Paint paint);`
> Sets the `Paint` used to draw the portion of the task that is not yet completed and sends a `RendererChangeEvent` to all registered listeners.

The width of the "percentage complete" indicator can be controlled by specifying the start and end percentage values relative to the width (not length!) of the task bars:

> `public void setStartPercent(double percent);`
> Sets the start position for the indicator as a percentage of the width of the task bar (for example, `0.30` is thirty percent)

> `public void setEndPercent(double percent);`
> Sets the end position for the indicator as a percentage of the width of the task bar (for example, `0.70` is seventy percent)

As an example, by setting the start and end percentages in the above methods to `0.30` and `0.70` (say), the middle forty percent of the task bar is occupied by the "percentage complete" indicator.

30.14.3 Notes

Some points to note:

- the `GanttDemo1.java` and `GanttDemo2.java` applications (included in the JFreeChart distribution) provide examples of this renderer being used.

30.15 HighLow

30.15.1 Overview

Represents one item used by a `HighLowRenderer` during the rendering process.

30.16 HighLowRenderer

30.16.1 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.

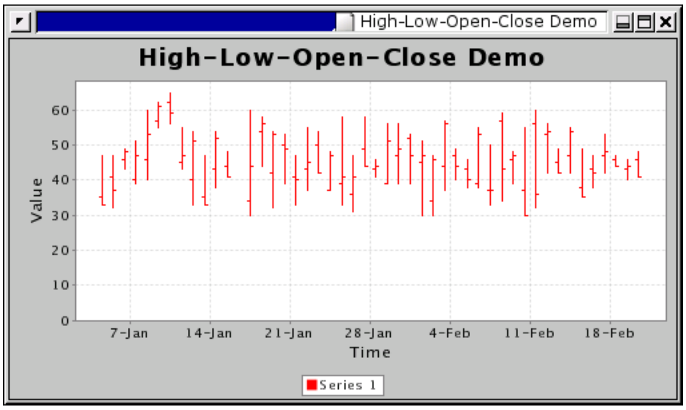


Figure 30.11: A chart that uses a `HighLowRenderer`

This renderer is designed for use with the `XYPlot` class. It requires a `HighLowDataset`.

30.16.2 Constructors

To create a new renderer:

```
public HighLowRenderer();
```
Creates a new renderer.

30.16.3 Methods

Implements the `drawItem()` method defined in the `XYItemRenderer` interface.

30.16.4 Notes

This renderer requires the dataset to be an instance of `HighLowDataset`.

The `createHighLowChart()` method in the `ChartFactory` class makes use of this renderer.

30.17 IntervalBarRenderer

30.17.1 Overview

A renderer that draws bars to represent items from an `IntervalCategoryDataset`.

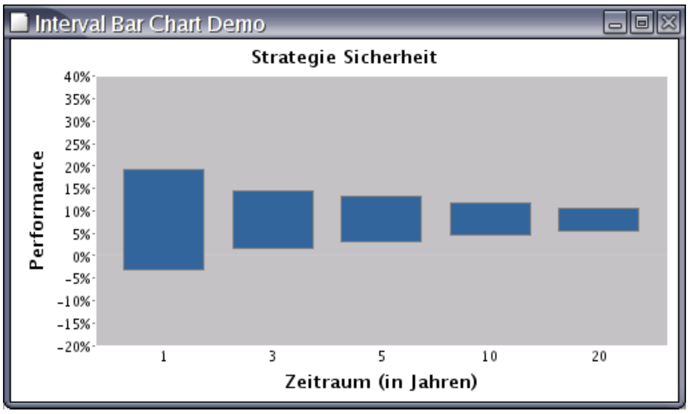


Figure 30.12: A chart that uses an `IntervalBarRenderer`

30.17.2 Notes

Some points to note:

- the `IntervalCategoryItemLabelGenerator` can be used to generate item labels (including tooltips) with this renderer.

See Also

`GanttRenderer`.

30.18 LevelRenderer

30.18.1 Overview

A renderer that draws horizontal lines to represent items from an `CategoryDataset`. The lines occupy the same width along the axis that a bar drawn by the `BarRenderer` class would occupy.

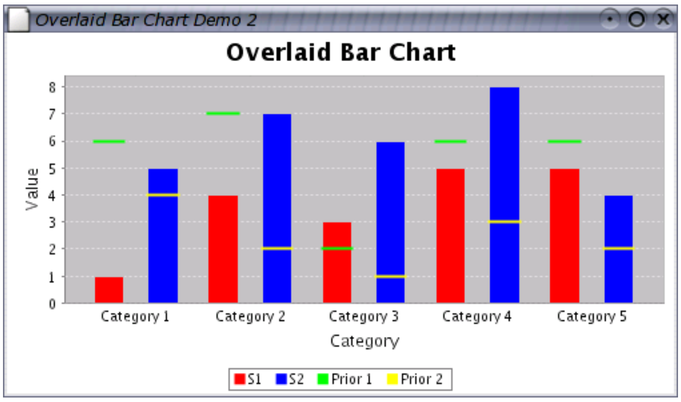


Figure 30.13: A chart that uses a `LevelRenderer`

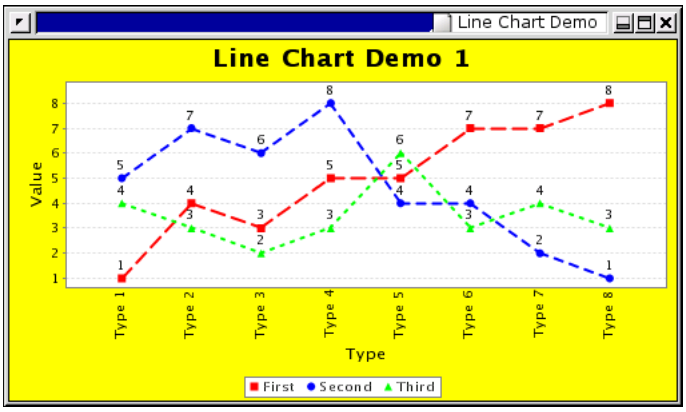
30.18.2 Notes

The `OverlaidBarChartDemo2` application (included in the JFreeChart distribution) provides a demo of this renderer.

30.19 LineAndShapeRenderer

30.19.1 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 `CategoryPlot` class.

30.19.2 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`.

30.19.3 Methods

To control the drawing of lines between data points:

> `public void setDrawLines(boolean draw);`
> Sets a flag that controls whether or not lines are drawn between data points. Notes that no line is drawn if a `null` data values is encountered.

To control the drawing of shapes at each data point:

> `public void setDrawShapes(boolean draw);`
> Sets the flag that controls whether or not shapes are drawn at each data point.

If shapes are drawn at each data point, you can set a flag that controls whether or not the shapes are filled. The following two methods allow you to specify the setting for ALL series:

> `public void setShapesFilled(boolean filled);`
> Sets a flag that controls whether or not shapes are filled for ALL series.

> `public void setShapesFilled(Boolean filled);`
> As above, but using a `Boolean` object. This allows the flag to be set to `null`, which means that the *per series* settings will apply.

This class implements the `drawCategoryItem(...)` method that is defined in the `CategoryItemRenderer` interface.

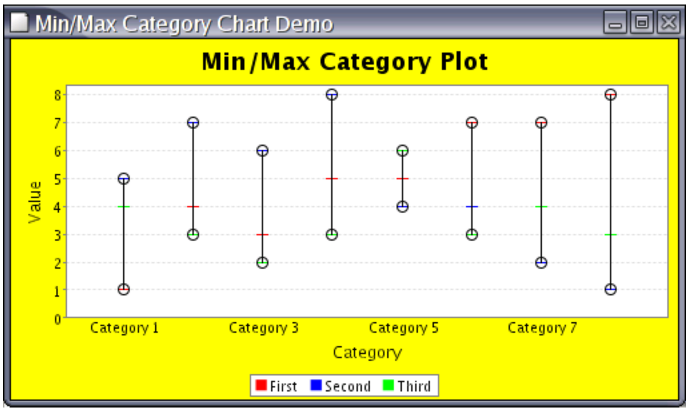
30.19.4 Equals, Cloning and Serialization

This renderer overrides the `equals()` method, and is `Cloneable` and `Serializable`. For general issues about these methods, refer to section 30.3.5.

30.20 MinMaxCategoryRenderer

30.20.1 Overview

A renderer that draws minimum and maximum markers.



30.21 NoOutlierException

30.21.1 Overview

An exception that can be generated by the classes used for creating box-and-whisker plots.

30.22 Outlier

30.22.1 Overview

Represents an outlier in a box-and-whisker plot.

30.23 OutlierList

30.23.1 Overview

Represents a collection of outliers for a single item in a box-and-whisker plot.

30.24 OutlierListCollection

30.24.1 Overview

Represents a collection of outlier lists for a box-and-whisker plot.

30.25 PolarItemRenderer

30.25.1 Overview

A renderer that is used by the `PolarPlot` class. The `DefaultPolarItemRenderer` class provides an implementation of this interface.

30.25.2 Change Listeners

You can register any number of `RendererChangeListener` objects with the renderer and they will receive notification of any changes to the renderer:

> `public void addChangeListener(RendererChangeListener listener);`
> Registers a listener with the renderer.

> `public void removeChangeListener(RendererChangeListener listener);`
> Deregisters a listener so that it no longer receives change notifications from the renderer.

30.25.3 Methods

To create a legend item for a series (this method is called by the plot):

> `public LegendItem getLegendItem(int series);`
> Creates a legend item for the specified series.

To draw the representation of a series:

> `public void drawSeries(...);`
> Renders the specified series.

30.26 RangeType

30.26.1 Overview

The *range type* relates to the way a renderer presents data, and is used when calculating the "auto-range" for an axis (that is, the range that will cause *all* the data to appear on a chart). There are two range types defined: `STANDARD` and `STACKED`.

In the standard case, a renderer just plots the values, so the maximum and minimum values in the dataset define the range of values.

An alternative treatment, used for example by the `StackedBarRenderer` class, is to stack values within a category. In this case, it is the maximum and minimum of the *sum of the values within a category* that determine the range of values.

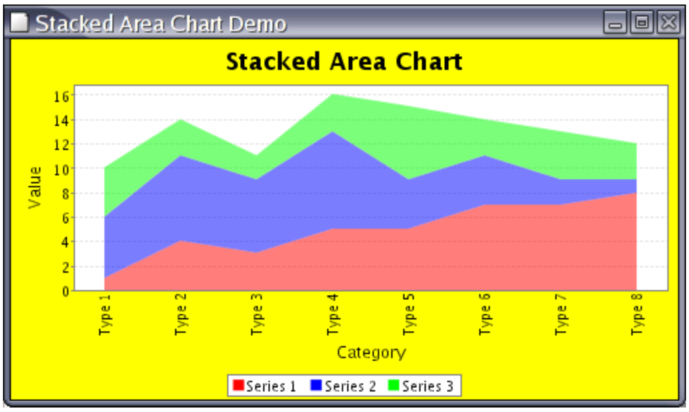
30.26.2 Notes

Every `CategoryItemRenderer` returns its range type via the `getRangeType()` method.

30.27 StackedAreaRenderer

30.27.1 Overview

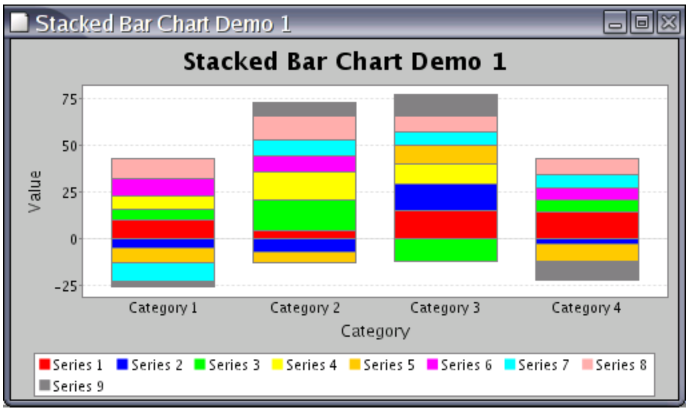
A stacked area renderer that draws items from a `CategoryDataset`.



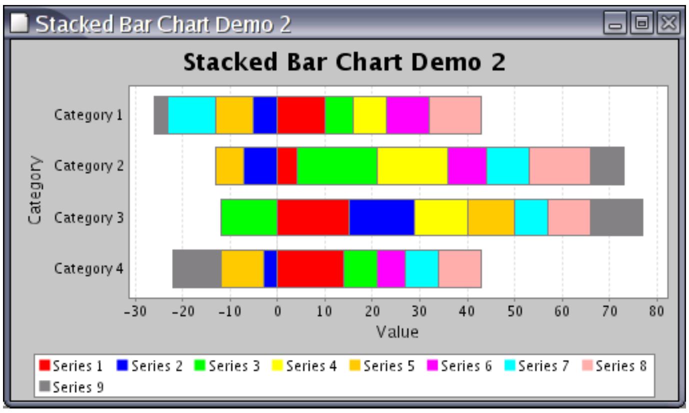
30.28 StackedBarRenderer

30.28.1 Overview

A *stacked bar renderer* draws each item in a `CategoryDataset` in the form of "stacked" bars. For example:



Here is another example, this time with a horizontal orientation:



This renderer is designed for use with the `CategoryPlot` class.

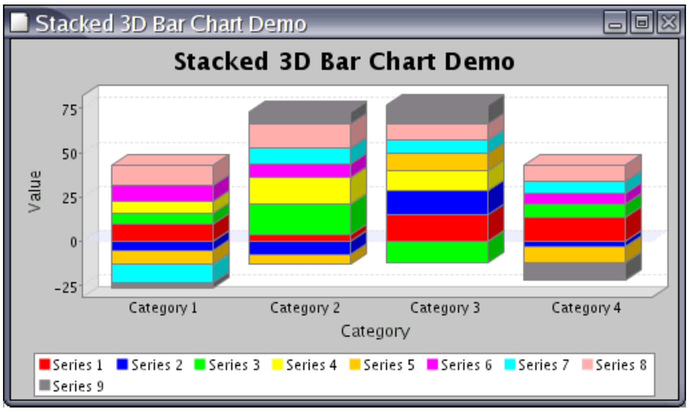
30.28.2 Methods

This class implements the methods in the CategoryItemRenderer interface.

30.29 StackedBarRenderer3D

30.29.1 Overview

A *stacked 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 CategoryPlot class.

30.29.2 Methods

This class implements the methods in the CategoryItemRenderer interface.

See Also

StackedBarRenderer.

30.30 StackedXYAreaRenderer

30.30.1 Overview

A stacked area renderer that draws items from a TableXYDataset. An example is shown in figure 30.14.

30.30.2 Notes

There is a demo (StackedXYAreaChartDemo1.java) that uses this renderer included in the JFreeChart distribution (in the src/org/jfree/chart/demo directory).

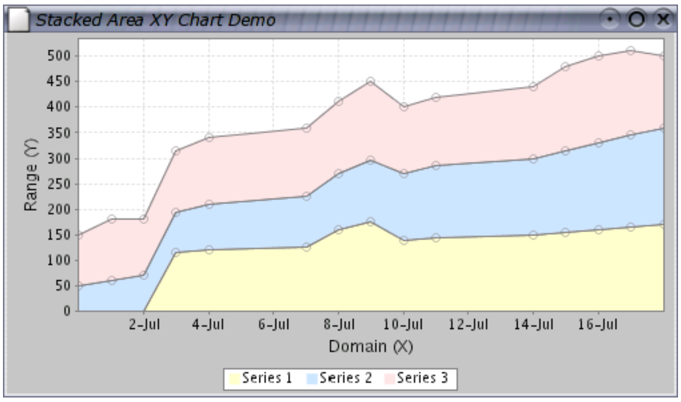


Figure 30.14: A chart created using `StackedXYAreaRenderer`

30.31 StandardXYItemRenderer

30.31.1 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).

30.31.2 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`.

30.31.3 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.

30.31.4 Notes

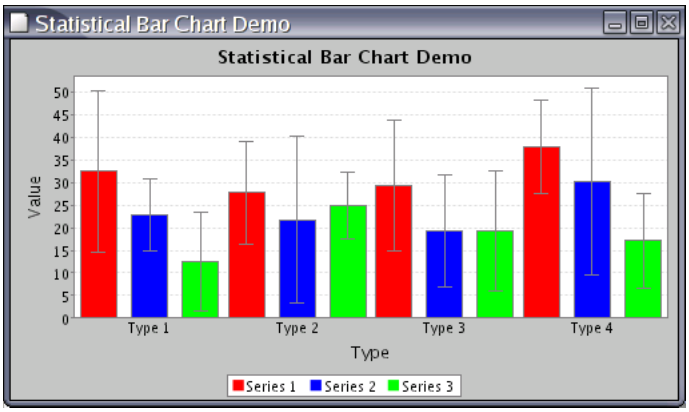
This class implements the `XYItemRenderer` interface.

The `XYPlot` class will use an instance of this class as its default renderer.

30.32 StatisticalBarRenderer

30.32.1 Overview

A *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 `CategoryPlot` class.

30.32.2 Notes

This class implements the `CategoryItemRenderer` interface.

30.33 WindItemRenderer

30.33.1 Overview

A renderer that `XYPlot` uses to draw wind plots.

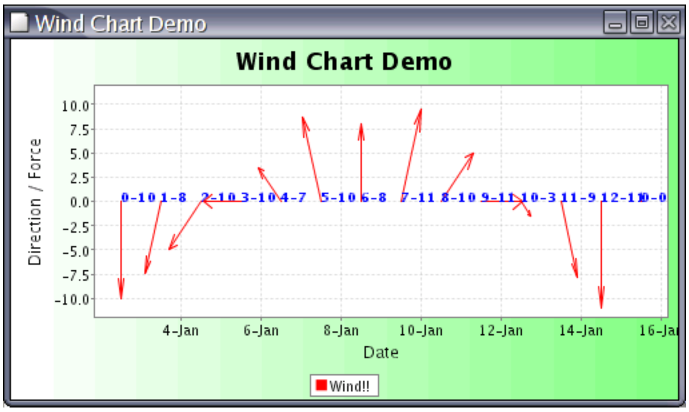
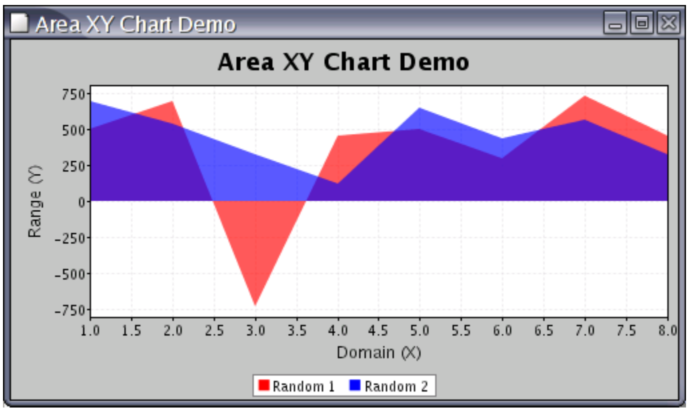


Figure 30.15: A sample chart using `WindItemRenderer`

30.34 XYAreaRenderer

30.34.1 Overview

An *XY area 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.

30.34.2 Constructors

The default constructor sets up the renderer to draw area charts:

> `public XYAreaRenderer();`
> Creates a new renderer.

You can change the appearance of the chart by specifying the type:

> `public XYAreaRenderer(int type);`
> Creates a new `XYAreaRenderer` using one of the following types: `SHAPES`, `LINES`, `SHAPES_AND_LINES`, `AREA`, `AREA_AND_SHAPES`.

30.34.3 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.

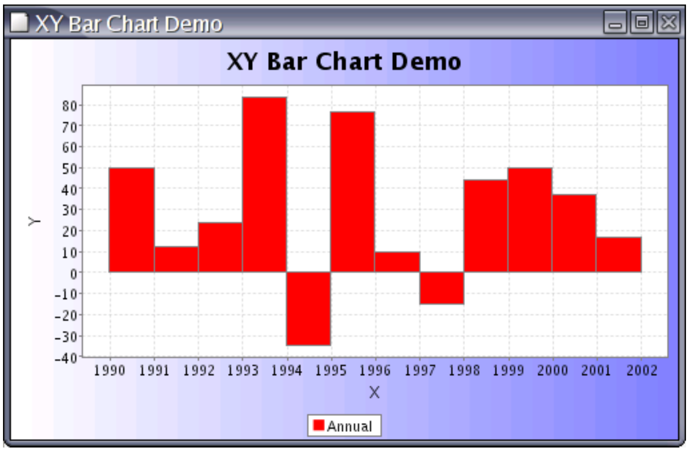
See Also

AreaRenderer.

30.35 XYBarRenderer

30.35.1 Overview

An *XY bar renderer* draws items from an IntervalXYDataset in the form of bars.



This renderer is designed to work with an XYPlot.

30.35.2 Constructors

The only constructor takes no arguments.

30.35.3 Methods

The drawItem(...) method handles the rendering of a single item for the plot.

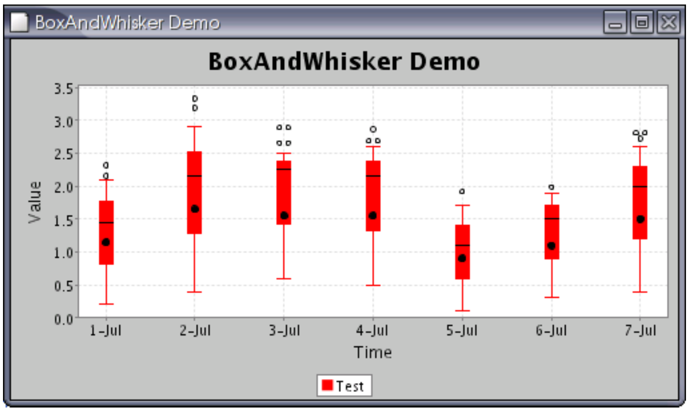
30.35.4 Notes

This renderer casts the dataset to IntervalXYDataset, so you should ensure that the plot is supplied with the correct type of data.

30.36 XYBoxAndWhiskerRenderer

30.36.1 Overview

A renderer that is used to create a *box-and-whisker plot*.



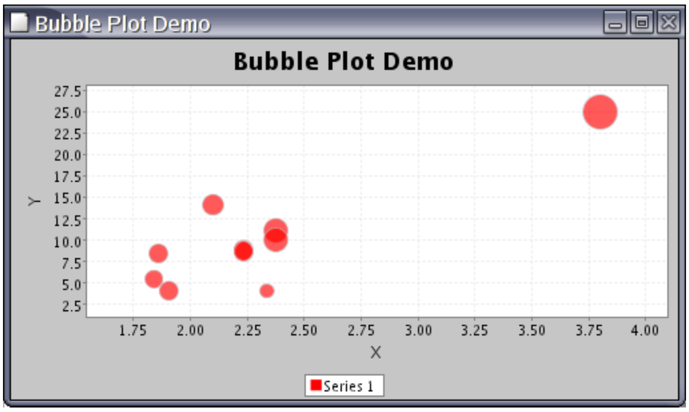
30.36.2 Notes

The `BoxAndWhiskerDemo` application, included in the JFreeChart distribution, provides an example of this type of plot.

30.37 XYBubbleRenderer

30.37.1 Overview

An *XY bubble renderer* displays items from an `XYZDataset` by drawing a bubble at each *(x, y)* point.



30.37.2 Notes

Some notes:

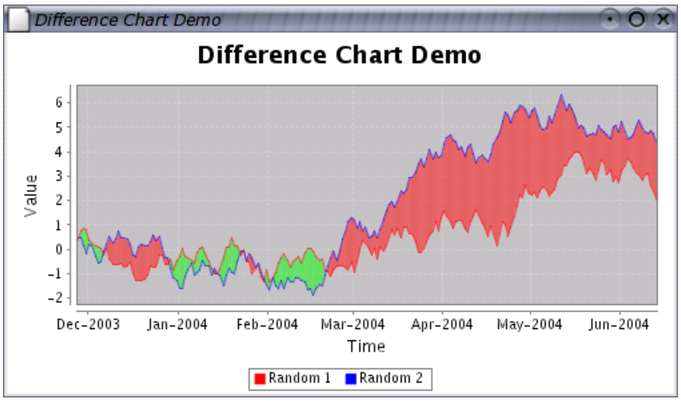
- this class implements the `XYItemRenderer` interface and extends the `AbstractXYItemRenderer` class.

- the `BubblePlotDemo` application (included in the JFreeChart distribution) provides a demonstration of this renderer.

30.38 XYDifferenceRenderer

30.38.1 Overview

A renderer that displays the difference between two series.



The `DifferenceChartDemo.java` application (included in the JFreeChart distribution) provides an example of this renderer being used.

30.39 XYDotRenderer

30.39.1 Overview

A renderer that can be used by an `XYPlot` to display items from an `XYDataset`. The renderer draws a pixel-sized dot at each *(x, y)* point—see figure 30.16 for an example.

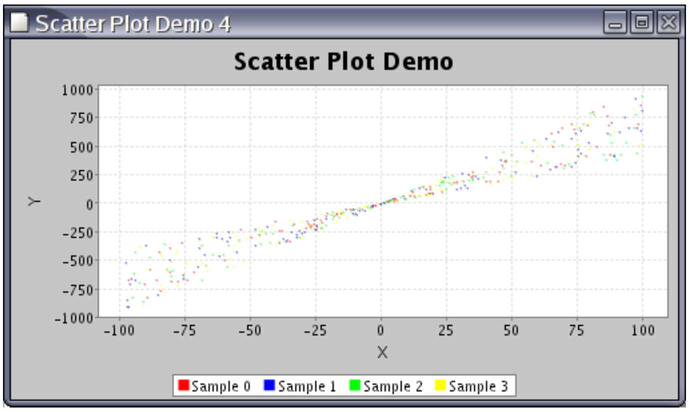


Figure 30.16: A chart generated with an `XYDotRenderer`*.*

This class implements the `XYItemRenderer` interface.

30.39.2 Constructor

The default constructor is the only constructor available:

> `public XYDotRenderer();`
> Creates a new renderer.

30.39.3 Methods

This class implements the `drawItem()` method defined in the `XYItemRenderer` interface. This method is usually called by the plot, you don't need to call it yourself. Many other methods are inherited from the `AbstractXYItemRenderer` base class.

30.39.4 Notes

Some points to note:

- this class extends the `AbstractXYItemRenderer` class;
- tooltips, item labels and URLs are NOT generated by this renderer (these features may be added in a future release);
- this class implements the `PublicCloneable` interface;
- instances of this class are `Serializable`;
- a demo application (`ScatterPlotDemo4.java`) is included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

30.40 XYItemRenderer

30.40.1 Overview

An *XY item renderer* is a plug-in class that works with an `XYPlot` and assumes responsibility for drawing individual data items in a chart. This interface defines the methods that every renderer must support.

A range of different renderers are supplied in the JFreeChart distribution. Figure 30.17 shows the class hierarchy.

As well as drawing the visual representation of a data item, the renderer is also responsible for generating tooltips (for charts displayed in a `ChartPanel`) and URL references for charts displayed in an HTML image map.

A summary of the available renderers is given in Table 30.4.

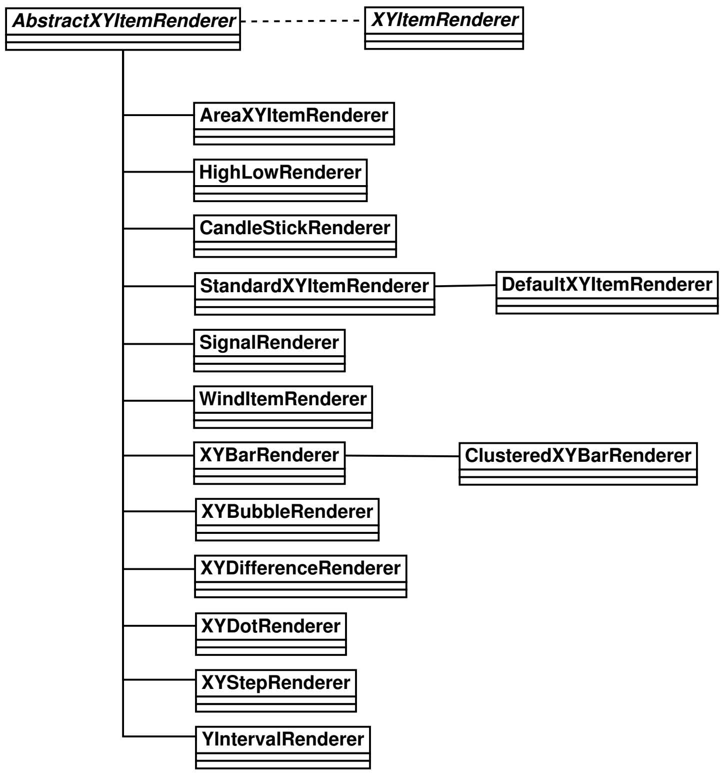


Figure 30.17: Renderer hierarchy

30.40.2 Methods

The `initialise()` method is called once at the beginning of the chart drawing process, and gives the renderer a chance to initialise itself:

```
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.

Class:	Description:
HighLowRenderer	High-low-open-close charts.
StandardXYItemRenderer	Line charts and scatter plots.
WindItemRenderer	Wind charts.
XYAreaRenderer	Area charts.
XYBarRenderer	Bar charts with numerical domain values.
XYBubbleRenderer	Bubble charts.
XYDifferenceRenderer	Difference charts.
XYDotRenderer	Scatter plots.
XYStepRenderer	Step charts.
YIntervalRenderer	Interval charts.

Table 30.4: Classes that implement the `XYItemRenderer` interface

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.

30.40.3 Notes

Some renderers require the a dataset that is a specific extension of `XYDataset`. For example, the `HighLowRenderer` requires a `HighLowDataset`.

See Also

`AbstractXYItemRenderer`, `XYPlot`.

30.41 XYLineAndShapeRenderer

30.41.1 Overview

A *renderer* that displays items from an `XYDataset` by drawing a line between each *(x, y)* point and overlaying a shape at each *(x, y)* point. One of the key features of this renderer is that it allows you to control on a *per series* basis whether:

- lines are drawn between the data points;
- shapes are drawn at each data point;
- shapes are filled or not filled;

This class implements the `XYItemRenderer` interface and extends the `AbstractXYItemRenderer` class.

30.41.2 Methods

To control whether or not lines are displayed for a series, use the following method:

```
public void setSeriesLinesVisible(int series, boolean visible);
```

Sets a flag that controls whether or not lines are visible for the specified series.

The flags are stored as `Boolean` objects—if the flag is `null` for a series, then the default value is returned. You can set the default value using:

```
public void setDefaultLinesVisible(boolean flag);
```

Sets the default flag that controls whether or not the renderer draws lines between the *(x, y)* items in a series.

It is recommended that you set the default value as required first, and then override the setting on a per series basis. If you have set the flag for a series, but later want to restore the default value, note that there is a version of the `setSeriesLinesVisible()` method that accepts a `Boolean` flag which you can set to `null`.

The settings that control whether or not shapes are drawn and filled follow a very similar pattern. There are default values that can be overridden on a *per series* basis.

30.41.3 Notes

Some points to note:

- the renderer makes two passes through the data. In the first pass, the lines connecting the *(x, y)* data points are drawn. In the second pass, the shapes at each data point are drawn. In this way, the lines appear to be "under" the shapes, which makes for a better presentation;
- there is some overlap between this class and the `StandardXYItemRenderer` class.

30.42 XYStepRenderer

30.42.1 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`

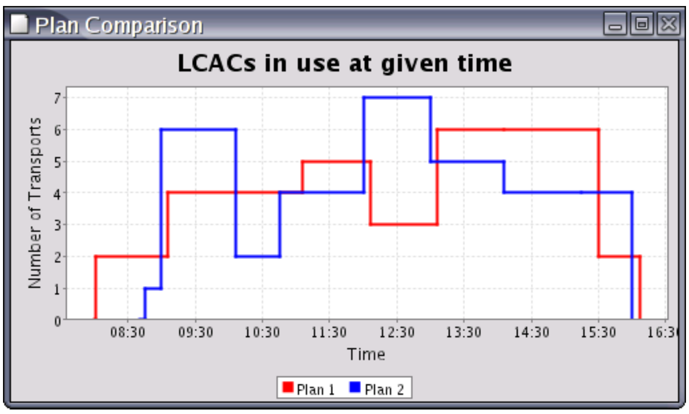


Figure 30.18: A sample chart using `XYStepRenderer`

class.

30.42.2 Usage

A demo (`XYStepChartDemo.java`) is included in the JFreeChart distribution (in the `src/org/jfree/chart/demo` directory).

30.43 YIntervalRenderer

30.43.1 Overview

An XYItemRenderer that draws lines between the starting and ending y values from an IntervalXYDataset.

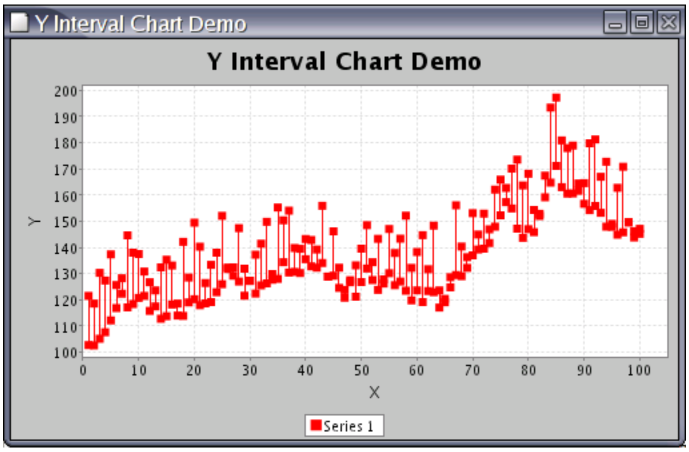


Figure 30.19: A sample chart using `YIntervalRenderer`

This renderer is designed for use with the XYPlot class.

30.43.2 Notes

The `YIntervalChartDemo` class in the `org.jfree.chart.demo` package provides an example of this renderer in use.

Chapter 31

Package: org.jfree.chart.servlet

31.1 Overview

This package contains servlet utility classes developed for JFreeChart by Richard Atkinson. An excellent demo for these classes can be found at:

`http://homepage.ntlworld.com/richard_c_atkinson/jfreechart`

31.2 ChartDeleter

31.2.1 Overview

A utility class that maintains a list of temporary files (chart images created by the `ServletUtilities` class) and deletes them at the expiry of an `HttpSession`.

31.3 DisplayChart

31.3.1 Overview

A servlet that displays a chart image from the temporary directory.

31.4 ServletUtilities

31.4.1 Overview

A utility class for performing operations in a servlet environment.

31.4.2 Methods

To save a chart in the temporary directory:

`public static String saveChartAsPNG(JFreeChart chart, int width, int height, ChartRenderingInfo info, HttpSession session);`

Saves a chart to a PNG image file in the temporary directory. The file is registered with a `ChartDeleter` instance that is linked to the specified session—this means the image file will be deleted when the session expires. Note that the temporary file name prefix can be set using the `setTempFilePrefix()` method.

Chapter 32

Package: org.jfree.chart.title

32.1 Overview

This package contains classes that are used as chart titles and/or subtitles. The `JFreeChart` class maintains one chart title (an instance of `TextTitle`) plus a list of subtitles (which can be any subclass of `Title`).

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.

32.2 Events

When you add a `Title` to a `JFreeChart` instance, the chart registers itself as a `TitleChangeListener`. Any subsequent changes to the title will result in a `TitleChangeEvent` being sent to the chart. The chart then passes the event on to all its registered `ChartChangeListener`s. If the chart is displayed in a `ChartPanel`, the panel will receive a `ChartChangeEvent` and respond by repainting the chart.

32.3 DateTitle

32.3.1 Overview

A chart title that displays the current date (extends `TextTitle`). This class would normally be used to add the date to a chart as a subtitle.

32.3.2 Constructor

To create a new date title for the default locale:

`public DateTitle(int style);`
Creates a new date title with the specified style (defined by the `DateFormat` class). The title position is, by default, the lower right corner of the chart.

32.3.3 Methods

To set the date format:

`public void setDateFormat(int style, Locale locale);`
Sets the date format to the given style and locale (the style is defined by constants in the `DateFormat` class).

Other methods are inherited from the `TextTitle` class.

32.4 ImageTitle

32.4.1 Overview

A chart title that displays an image (extends `Title`).

32.4.2 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.

Methods

To change the image displayed by the image title:

`public void setImage(Image image);`
Sets the image for the title and sends a `TitleChangeEvent` to all registered listeners.

Other methods are inherited from the `Title` class.

32.5 LegendTitle

32.5.1 Overview

This class is ultimately intended to make the legend behave in the same way as all other chart titles, but is currently incomplete.

32.6 TextTitle

32.6.1 Overview

A chart title that displays a text string (extends `Title`).

32.6.2 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`.

32.6.3 Methods

To set the title string:

> `public void setText(String text);`
> Sets the text for the title and sends a `TitleChangeEvent` to all registered listeners.

To set the font for the title:

> `public void setFont(Font font);`
> Sets the font for the title and sends a `TitleChangeEvent` to all registered listeners.

To set the color of the title:

> `public void setPaint(Paint paint);`
> Sets the paint used to display the title text and sends a `TitleChangeEvent` to all registered listeners.

The following method is called by the `JFreeChart` class to 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 `Title` class.

32.6.4 Notes

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`

32.7 Title

32.7.1 Overview

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

32.7.2 Constructors

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

> `protected Title(RectangleEdge position,`
> `HorizontalAlignment horizontalAlignment, VerticalAlignment verticalAlignment,`
> `Spacer spacer);`
> Creates a new `Title` with the specified position, alignment and spacing.

32.7.3 Methods

You can set the position for a title using the `RectangleEdge` class, which defines an enumeration with the values `TOP`, `BOTTOM`, `LEFT` and `RIGHT`:

> `public void setPosition(RectangleEdge position);`
> Sets the position for the title (`null` not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners (the `JFreeChart` object that the title belongs to is registered by default).

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

> `public void setHorizontalAlignment(HorizontalAlignment alignment);`
> Sets the horizontal alignment for the title (`null` not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners.

Similarly, you can specify the vertical alignment:

> `public void setVerticalAlignment(VerticalAlignment alignment);`
> Sets the vertical alignment for the title (`null` not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners.

To control the space around the outside of the title, you can use a `Spacer`:

> `public void setSpacer(Spacer spacer);`
> Sets the spacer object for the title and sends a `TitleChangeEvent` to all registered listeners.

32.7.4 Notes

Some points to note:

- 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.

Chapter 33

Package: org.jfree.chart.ui

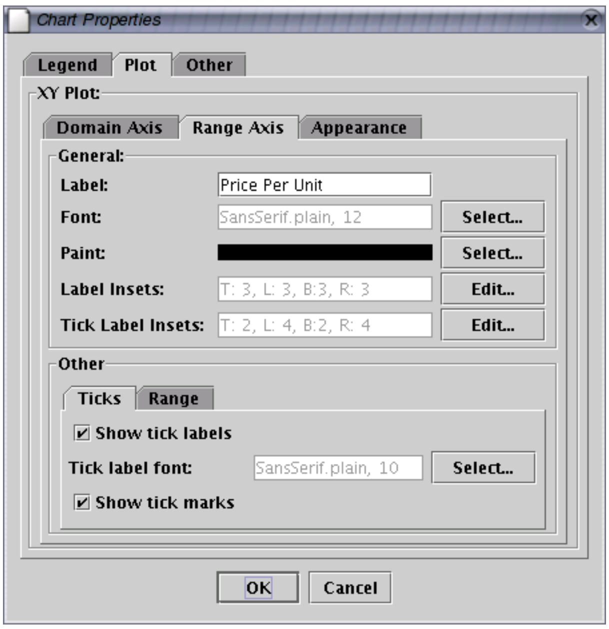
33.1 Introduction

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.

33.1.1 AxisPropertyEditPanel

33.1.2 Overview

A panel for editing the properties of an axis.



The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite the property editors before JFreeChart 1.0.0 is released.

33.2 ChartPropertyEditPanel

33.2.1 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 `TitlePropertyEditPanel`;
- a `LegendPropertyEditPanel`;
- a `PlotPropertyEditPanel`;
- 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 `TitlePropertyEditPanel` and is used if and when a sub-dialog is required for editing titles.

33.3 ColorBarPropertyEditPanel

33.3.1 Overview

A panel for editing the properties of a `ColorBar`.

33.4 ColorPalette

33.4.1 Overview

The abstract base class for the color palettes used by the `ContourPlot` class.

33.5 GreyPalette

33.5.1 Overview

A grey palette (extends `ColorPalette`).

33.6 LegendPropertyEditPanel

33.6.1 Overview

A panel for displaying and editing the properties of a chart legend.
The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite the property editors before JFreeChart 1.0.0 is released.

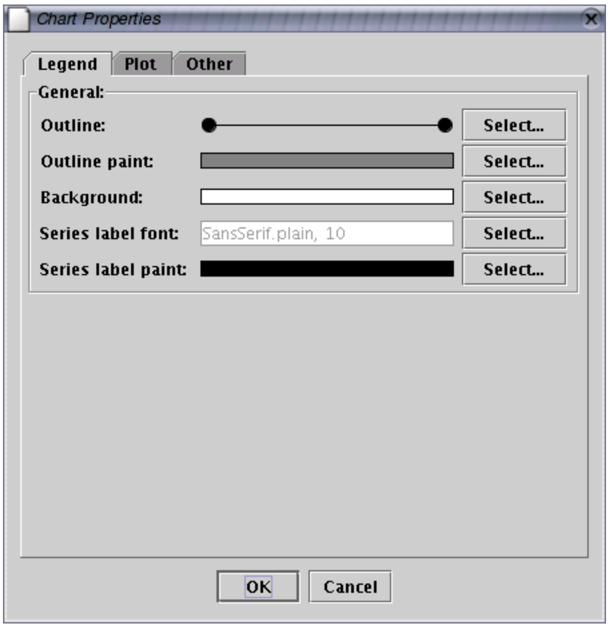


Figure 33.1: The legend property editor

33.7 NumberAxisPropertyEditPanel

33.7.1 Overview

A panel for displaying and editing the properties of a `NumberAxis`.

33.8 PaletteChooserPanel

33.8.1 Overview

A panel for selecting a color palette.

33.9 PlotPropertyEditPanel

33.9.1 Overview

A panel for displaying and editing the properties of a plot.
The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite the property editors before JFreeChart 1.0.0 is released.

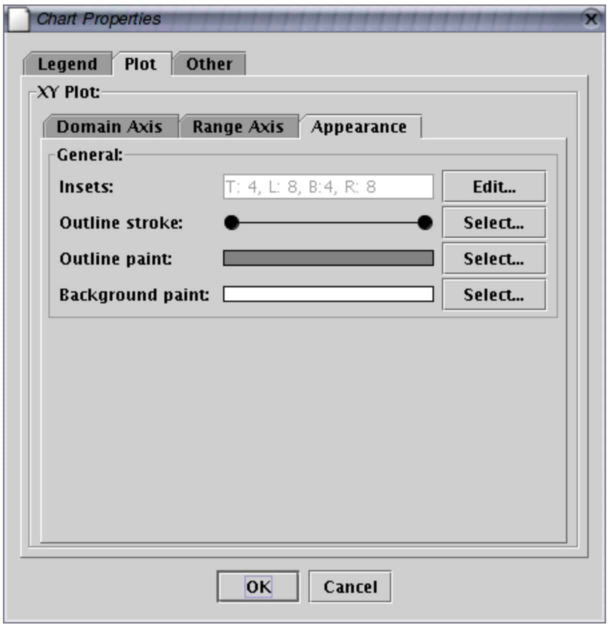


Figure 33.2: The plot property editor

33.10 RainbowPalette

33.10.1 Overview

A rainbow palette (extends ColorPalette).

33.11 TitlePropertyEditPanel

33.11.1 Overview

A panel for displaying and editing the properties of a chart title. The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite the property editors before JFreeChart 1.0.0 is released.

Chapter 34

Package: org.jfree.chart.urls

34.1 Overview

This package contains support for URL generation for HTML image maps. URLs are generated (if they are required) at the point that a renderer draws the visual representation of a data item. The renderer queries a *URL generator* via one of the following interfaces:

- `CategoryURLGenerator`;
- `PieURLGenerator`;
- `XYURLGenerator`;
- `XYZURLGenerator`;

JFreeChart provides standard implementations for each of these interfaces. In addition, you can easily write your own implementation and take full control of the URLs that are generated within your image map.

34.2 CategoryURLGenerator

34.2.1 Overview

A *category URL generator* is used to generate a URL for each data item in a `CategoryPlot`. The generator is associated with the plot's renderer (an instance of `CategoryItemRenderer`) and the URLs are used when you create an HTML image map for a chart image.

34.2.2 Methods

This method returns a URL for a specific data item:

> `public String generateURL(CategoryDataset data, int series, int category);`
> Returns a URL for the specified data item. The `series` is the row index, and the `category` is the column index for the dataset.

34.2.3 Notes

Some points to note:

- the `StandardCategoryURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library, but you can add your own implementation(s);
- the `ChartUtilities` class contains code for writing HTML image maps.

34.3 CustomXYURLGenerator

34.3.1 Overview

A URL generator that uses custom strings as the URL for each item in an `XYDataset`. This class implements the `XYURLGenerator` interface.

34.4 PieURLGenerator

34.4.1 Overview

A *pie URL generator* is used by a `PiePlot` to generate URLs for use in HTML image maps.

34.4.2 Methods

This method returns a URL for a specific data item:

> `public String generateURL(PieDataset dataset, Comparable key, int pieIndex);`
> Returns a URL for the specified data item. The `key` is the key for the current section within the dataset, and the `pieIndex` is used when multiple pie plots are included within one chart.

34.4.3 Notes

Some points to note:

- the `StandardPieURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

34.5 StandardCategoryURLGenerator

34.5.1 Overview

A class that generates a URL for a data item in a `CategoryPlot`. By default, this generator will create URLs in the format:

```
index.html?series=<serieskey>&category=<categorykey>
```

This class implements the `CategoryURLGenerator` interface.

34.5.2 Usage

If you create a chart using the ChartFactory class, you can ask for a default URL generator to be installed in the renderer just by setting the urls flag (a parameter for most chart creation methods) to true.

Alternatively, you can create a new generator and register it with the renderer (replacing the existing generator, if there is one) as follows:

```
CategoryPlot plot = chart.getCategoryPlot();
CategoryItemRenderer renderer = plot.getRenderer();
CategoryURLGenerator generator = new StandardCategoryURLGenerator(
    ``index.html'',
    ``series'',
    ``category''
);
renderer.setItemURLGenerator(generator);
```

Set the URL generator to null if you do not require URLs to be generated.

34.5.3 Constructors

To create a new generator:

> `public StandardCategoryURLGenerator(String prefix,`
> `String seriesParameterName, String categoryParameterName);`
> Creates a new generator with the specified attributes.

34.5.4 Methods

The following method is called by the renderer to generator the URL for a single data item in a chart:

> `public String generateURL(CategoryDataset data, int series, int category)`
> Returns a string that will be used as the URL for the specified data item.

34.5.5 Notes

Some points to note:

- this class is the only implementation of the CategoryURLGenerator interface that is provided by JFreeChart, but you can easily write your own implementation.

34.6 StandardPieURLGenerator

34.6.1 Overview

A default URL generator for use when creating HTML image maps for pie charts. This class implements the PieURLGenerator interface.

34.6.2 Constructor

To create a new generator:

> `public StandardPieURLGenerator(String prefix, String categoryParameterName);`
> Creates a new generator.

34.7 StandardXYURLGenerator

34.7.1 Overview

A default URL generator for creating HTML image maps. This class implements the `XYURLGenerator` interface.

34.8 StandardXYZURLGenerator

34.8.1 Overview

A URL generator that creates URLs for the items in an `XYZDataset`.

34.9 TimeSeriesURLGenerator

34.9.1 Overview

A URL generator that creates URLs for the items in an `XYDataset`. The x-values from the dataset are evaluated as "milliseconds since midnight 1-Jan-1970" (as for `java.util.Date`) and converted to date format.

34.10 XYURLGenerator

34.10.1 Overview

An *XY URL generator* is used by a `XYItemRenderer` to generate URLs for use in HTML image maps.

34.10.2 Methods

This method returns a URL for a specific data item:

```
public String generateURL(XYDataset data, int series, int item);
```

Returns a URL for the specified data item.

34.10.3 Notes

Some points to note:

- the `StandardXYURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

34.11 XYZURLGenerator

34.11.1 Overview

An *XYZ URL generator* is used by a `XYItemRenderer` to generate URLs for use in HTML image maps.

34.11.2 Methods

This method returns a URL for a specific data item:

```
public String generateURL(XYDataset data, int series, int item);
```

Returns a URL for the specified data item.

34.11.3 Notes

Some points to note:

- the `StandardXYURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

Chapter 35

Package: org.jfree.data

35.1 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.

35.2 AbstractDataset

35.2.1 Overview

A useful base class for implementing the `Dataset` interface (or extensions). This class provides a default implementation of the *change listener* mechanism.

35.2.2 Constructors

The default constructor:

> `protected AbstractDataset();`
> Allocates storage for the registered change listeners.

35.2.3 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.

35.2.4 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`.

35.3 AbstractSeriesDataset

35.3.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

35.3.2 Constructors

This class is never instantiated directly, so the constructor is protected:

```
protected AbstractSeriesDataset();
```

Simply calls the constructor of the superclass.

35.3.3 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.

35.3.4 Notes

This class is provided simply for convenience, you are not required to use it when developing your own dataset classes.

See Also

`Dataset`.

35.4 CategoryDataset

35.4.1 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 extends the `KeyedValues2D` and `Dataset` interfaces.

35.4.2 Methods

This interface adds no additional methods to those defined in the `KeyedValues2D` and `Dataset` interfaces.

35.4.3 Notes

Some points to note:

- 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.
- a useful implementation of this interface is provided by the `DefaultCategoryDataset` class.
- the `CategoryToPieDataset` class converts one row or column of the dataset into a `PieDataset`.
- you can read a `CategoryDataset` from a file (in a prespecified XML format) using the `DatasetReader` class.

See Also

`CategoryPlot`.

35.5 CategoryToPieDataset

35.5.1 Overview

A utility class that presents one row or column of data from a `CategoryDataset` via the `PieDataset` interface.

35.5.2 Constructor

To create a new instance:

```
public CategoryToPieDataset(CategoryDataset source, TableOrder extract,
int index);
```

Creates a new pie dataset based on the `source`. The `extract` argument specifies whether the dataset uses a row or column from the source dataset (use `TableOrder.BY_ROW` or `TableOrder.BY_COLUMN`), and the `index` controls which row or column is selected.

35.5.3 Notes

This class registers itself with the underlying `CategoryDataset` to receive change events. Whenever the underlying dataset is changed, a new `DatasetChangeEvent` is triggered and sent to all registered listeners.

35.6 CombinationDataset

35.6.1 Overview

An interface that defines the methods that should be implemented by a *combination dataset*.

35.6.2 Notes

This interface is implemented by the `CombinedDataset` class.

35.7 CombinedDataset

35.7.1 Overview

A dataset that can combine other datasets.

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`.

35.8 ContourDataset

35.8.1 Overview

The dataset used by the `ContourPlot` class.

35.8.2 Methods

This interface defines the following methods in addition to those inherited from the `XYZDataset` interface:

`public double getMinZValue();`
Returns the minimum z-value.

`public double getMaxZValue();`
Returns the maximum z-value.

`public Number[] getXValues();`
Returns an array containing all the x-values.

`public Number[] getYValues();`
Returns an array containing all the y-values.

`public Number[] getZValues();`
Returns an array containing all the z-values.

`public int[] indexX();`
Returns the index values.

`public int[] getXIndices();`
Returns an int array contain the index into the x values.

`public Range getZValueRange(Range x, Range y);`
Returns the maximum z-value for the specified visible region of the plot.

`public boolean isDateAxis(int axisNumber);`
Returns true if the values for the specified axis are dates (where `axisNumber` is defined as 0-x, 1-y, and 2-z).

See Also

`DefaultContourDataset`.

35.9 Dataset

35.9.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by `PieDataset`, `CategoryDataset` and `SeriesDataset`.

35.9.2 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.

35.9.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as `PieDataset`, `CategoryDataset` or `XYDataset`.

35.10 DatasetChangeEvent

35.10.1 Overview

An event that is used to provide information about changes to datasets.

35.10.2 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.

35.10.3 Methods

To get a reference to the `Dataset` that generated the event:

> `public Dataset getDataset();`
> Returns the dataset which generated the event.

35.10.4 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`.

35.11 DatasetChangeListener

35.11.1 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.

35.11.2 Methods

The interface defines a single method:

> `public void datasetChanged(DatasetChangeEvent event);`
> Receives notification of a change to a dataset.

35.11.3 Notes

In JFreeChart, the `Plot` class implements this interface in order to receive notification of changes to the dataset.

See Also

`DatasetChangeEvent`.

35.12 DatasetGroup

35.12.1 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.

35.13 DatasetUtilities

35.13.1 Overview

A collection of utility methods for working with datasets.

35.13.2 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.

To get the minimum "stacked" range value in a `CategoryDataset`:

> `public static Number getMinimumStackedRangeValue(CategoryDataset data);`
> Returns the minimum stacked range value in a dataset.

To get the maximum "stacked" range value in a `CategoryDataset`:

> `public static Number getMaximumStackedRangeValue(CategoryDataset data);`
> Returns the maximum stacked range value in a dataset.

35.13.3 Creating Datasets

To create a `PieDataset` from the data in one column of a `CategoryDataset`:

> `public static PieDataset createPieDatasetForColumn(CategoryDataset data, Comparable columnKey);`
> Returns a pie dataset by taking all the values in the category dataset for the specified column.

To create a `PieDataset` from the data in one row of a `CategoryDataset`:

> `public static PieDataset createPieDatasetForRow(CategoryDataset data, Comparable rowKey);`
> 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`.

35.14 DataUtilities

35.14.1 Overview

This class contains utility methods that relate to general data classes.

35.14.2 Methods

To calculate the cumulative percentage values from a collection of data values:

> `public static KeyedValues getCumulativePercentages(KeyedValues data);`
> Returns a new collection of data values containing the cumulative percentage values from the specified data.

35.15 DateRange

35.15.1 Overview

An extension of the `Range` class that is used to represent a date/time range. In JFreeChart, the primary use for this class is for specifying the range of values to display on a `DateAxis`.

35.15.2 Constructors

To create a new date range:

> `public DateRange(Date lower, Date upper);`
> Creates a new date range using the specified lower and upper bounds (do not use `null` for either parameter).

35.15.3 Notes

Instances of this class are immutable and `Serializable`.

35.16 DefaultCategoryDataset

35.16.1 Overview

A default implementation of the `CategoryDataset` interface.

35.16.2 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.

35.16.3 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.

35.16.4 Notes

This class uses an instance of `DefaultKeyedValues2D` to store its data.

35.17 DefaultContourDataset

35.17.1 Overview

A default implementation of the `ContourDataset` interface.

See Also

ContourPlot

35.18 DefaultHighLowDataset

35.18.1 Overview

A default implementation of the `HighLowDataset` interface.

35.19 DefaultIntervalCategoryDataset

35.19.1 Overview

A default implementation of the `IntervalCategoryDataset` interface.

35.20 DefaultKeyedValue

35.20.1 Overview

A *(key, value)* data item, where the key is an instance of `Comparable` and the value is an instance of `Number`. For the value, you can use `null` to represent a missing or unknown value. This class provides a default implementation of the `KeyedValue` interface.

35.20.2 Usage

This class is typically used to represent individual data items in a larger collection, such as `DefaultKeyedValues`.

35.20.3 Constructor

To create a new instance:

> `public DefaultKeyedValue(Comparable key, Number value);`
> Creates a new data item that associates a value with a key. The key should be an immutable object such as `String`. The value can be any `Number` instance, or `null` to represent a missing or unknown value.

35.20.4 Methods

There are methods to access the key and value attributes:

> `public Comparable getKey();`
> Returns the key.

> `public Number getValue();`
> Returns the value (possibly `null`).

Once a `DefaultKeyedValue` instance is created, the key can never be changed, but you can update the value:

`public synchronized void setValue(Number value);`
Sets the value for this data item.

35.20.5 Notes

Some points to note:

- cloning is supported, but no deep cloning is performed because it is assumed that both the key and value are immutable (we know this is true for the value, and assume it to be true for the key).
- this class is serializable provided that the key is serializable.

35.21 DefaultKeyedValueDataset

35.21.1 Overview

A dataset that contains a single *(key, value)* data item. This class implements the `KeyedValueDataset` interface.

35.21.2 Usage

This class does not get used by JFreeChart.

35.22 DefaultKeyedValues

35.22.1 Overview

A collection of *(key, value)* data items, where the key is an instance of `Comparable` and the value is an instance of `Number`.

35.22.2 Notes

Some points to note:

- this class provides a default implementation of the `KeyedValues` interface;
- the `DefaultPieDataset` class uses an instance of this class to store its data.

35.23 DefaultKeyedValuesDataset

35.23.1 Overview

A dataset that implements the `KeyedValuesDataset` interface.

35.23.2 Notes

This dataset extends the `DefaultPieDataset` class without modification—it exists for completeness sake, to follow the naming pattern established for related classes and interfaces.

35.24 DefaultKeyedValues2D

35.24.1 Overview

A storage structure for a table of values that are associated with keys. This class provides a default implementation of the KeyedValues2D interface.

35.24.2 Notes

The DefaultCategoryDataset class uses an instance of this class to store its data.

35.25 DefaultKeyedValues2DDataset

35.25.1 Overview

A default implementation of the KeyedValues2DDataset interface.

35.26 DefaultMeterDataset

35.26.1 Overview

A default implementation of the MeterDataset interface.

35.27 DefaultPieDataset

35.27.1 Overview

A default implementation of the PieDataset interface.

35.27.2 Constructors

To create a new pie dataset:

```
public DefaultPieDataset();
```

Creates a new dataset, initially empty.

35.27.3 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.

35.27.4 Notes

The dataset can contain `null` values.

See Also

`PiePlot`.

35.28 DefaultValueDataset

35.28.1 Overview

A default implementation of the `ValueDataset` interface.

35.29 DefaultWindDataset

35.29.1 Overview

A default implementation of the `WindDataset` interface.

35.30 DomainInfo

35.30.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's domain.

35.30.2 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.

35.30.3 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`.

35.31 Function2D

35.31.1 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.

35.31.2 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.

35.31.3 Notes

The `DatasetUtilities` class provides a method for creating an `XYDataset` by sampling the values of a function.

See Also

`LineFunction2D`, `PowerFunction2D`.

35.32 HighLowDataset

35.32.1 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.

35.32.2 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.

35.32.3 Notes

This dataset is implemented by the `DefaultHighLowDataset` class, and used by the `CandlestickRenderer` class.

See Also
`XYDataset`, `DefaultHighLowDataset`.

35.33 IntervalCategoryDataset

35.33.1 Overview

An extension of the `CategoryDataset` interface that adds methods for returning a *start value* and an *end value* for each item in the dataset.

Like a `CategoryDataset`, this dataset is conceptually a table of data items where the "categories" represent columns and the "series" represent rows. The cells within the table contain three items: the start value, the end value and the value (the final item may be the same as one of the previous values or it may be different).

35.33.2 Methods

To get the start value for a data item:

`public Number getStartValue(int series, int category);`
Returns the start value for the specified data item.

`public Number getStartValue(Comparable series, Comparable category);`
Returns the start value for the specified data item

To get the end value for a data item:

`public Number getEndValue(int series, int category);`
Returns the end value for the specified data item.

`public Number getEndValue(Comparable series, Comparable category);`
Returns the end value for the specified data item.

Note that all of the above methods can return `null` to represent a missing or unknown value.

35.33.3 Notes

Some points to note:

- the `IntervalBarRenderer` class expects to receive data from a dataset that implements this interface;
- the `DefaultIntervalCategoryDataset` class provides one implementation of this interface;

35.34 IntervalXYDataset

35.34.1 Overview

A dataset that returns an interval for each of the x and y dimensions. Extends the `XYDataset` interface.

35.34.2 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.

35.34.3 Notes

The `TimeSeriesCollection` class implements this interface.

See Also:
`XYDataset`, `IntervalXYZDataset`.

35.35 IntervalXYZDataset

35.35.1 Overview

An extension of the `XYZDataset` interface, analogous to the `IntervalXYDataset` extension of the `XYDataset` interface.

35.35.2 Notes

There are no classes that implement this interface at present.

35.36 JDBCCategoryDataset

35.36.1 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.

35.36.2 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.

35.36.3 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.

35.36.4 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`.

35.37 JDBCPieDataset

35.37.1 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.

35.37.2 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.

35.37.3 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.

35.37.4 Notes

There is a demo application `JDBCPieChartDemo` in the JFreeChart distribution that illustrates the use of this class.

See Also

`PieDataset`, `DefaultPieDataset`.

35.38 JDBCXYDataset

35.38.1 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.

35.38.2 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.

35.38.3 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.

35.38.4 Notes

There is a demo application `JDBCXYChartDemo` in the JFreeChart distribution that illustrates the use of this class.

See Also

`XYDataset`.

35.39 KeyedObject

35.39.1 Overview

Not yet documented.

35.40 KeyedObjects

35.40.1 Overview

Not yet documented.

35.41 KeyedObjects2D

35.41.1 Overview

Not yet documented.

35.42 KeyedValue

35.42.1 Overview

A *keyed value* is a value (`Number`) that is associated with a key (`Comparable`).

35.42.2 Methods

This interface extends the `Value` interface.

To access the key associated with the value:

> `public Comparable getKey();`
> Returns the key associated with the value.

35.42.3 Notes

The `DefaultKeyedValue` class provides one implementation of this interface.

35.43 KeyedValueComparator

35.43.1 Overview

This class is used to compare two `KeyedValue` objects, either by key or by value.

35.44 KeyedValueComparatorType

35.44.1 Overview

Used to represent the two comparison types—*by key* or *by value*—used by the `KeyedValueComparator` class.

35.45 KeyedValueDataset

35.45.1 Overview

A dataset that contains a single *(key, value)* data item, where the key is an instance of `Comparable` and the value is an instance of `Number`.

35.45.2 Methods

This interface extends the `KeyedValue` and `Dataset` interfaces, and adds no additional methods.

35.45.3 Notes

There are currently no charts that specifically require this type of dataset.

35.46 KeyedValues

35.46.1 Overview

A collection of *(key, value)* data items, where the key is an instance of `Comparable` and the value is an instance of `Number`. This interface extends the `Values` interface.

35.46.2 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.

35.46.3 Notes

Some points to note:

- the *(key, value)* pairs in the collection have a specific order, since each key is associated with a zero-based index;
- the `DefaultKeyedValues` class provides one implementation of this interface.

35.47 KeyedValuesDataset

35.47.1 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.

35.47.2 Methods

This interface adds no methods to those it inherits from the `KeyedValues` and `Dataset` interfaces.

35.48 KeyedValues2D

35.48.1 Overview

A table of values that can be accessed using a *row key* and a *column key*. This interface extends the `Values2D` interface.

35.48.2 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.

35.48.3 Notes

The `DefaultKeyedValues2D` class provides one implementation of this interface.

35.49 KeyedValues2DDataset

35.49.1 Overview

Equivalent to the `CategoryDataset` interface.

35.50 LineFunction2D

35.50.1 Overview

A simple function of the form $y = a + bx$.

35.50.2 Constructor

To construct a new line function:

`public LineFunction2D(double a, double b);`
Creates a new line function with the given coefficients.

35.50.3 Methods

`public double getValue(double x);`
Returns the value of the function for a given input.

35.50.4 Notes

This class implements the `Function2D` interface.

The `RegressionDemo1` application provides an example of this class being used.

See Also

`PowerFunction2D`.

35.51 MeanAndStandardDeviation

35.51.1 Overview

A simple class that records the mean and standard deviation for some data.

35.51.2 Notes

Used in the `DefaultStatisticalCategoryDataset` implementation.

35.52 MeterDataset

35.52.1 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.

35.52.2 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):

`public Number getMinimumCriticalValue();`
Returns the lower bound of the "critical" range.

`public Number getMaximumCriticalValue();`
Returns the upper bound of the "critical" range.

There is a method to query whether or not the current value is valid:

`public boolean isValueValid();`
Returns `true` if the current value is valid, and `false` otherwise.

Note that this method is redundant, since the `getValue()` method can just return `null` if there is no value available.

To get a description of the unit of measurement for the value returned by the dataset:

`public String getUnits();`
Returns a description of the unit of measurement for the dataset value.

One final method returns a "border type":

`public int getBorderType();`
Returns the border type. This should return one of: `NORMAL_DATA`, `WARNING_DATA`, `CRITICAL_DATA` and `FULL_DATA`.

I'm not sure of the purpose of this method, in the `MeterPlot` class it seems to change the color-coding of the subranges, but only a return value of `FULL_DATA` appears to make sense.

35.52.3 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`.

35.53 MovingAverage

35.53.1 Overview

A utility class for calculating a *moving average* for a data series (usually a `TimeSeries`). Moving averages are most commonly used in the analysis of stock prices or other financial data.

35.53.2 An Example

An example is perhaps the best way to illustrate how moving averages are calculated. A sample dataset containing daily data and a corresponding three-day moving average is presented in Table 35.1.

Date:	Value:	3 Day Moving Average:
11-Aug-2003	11.2	-
13-Aug-2003	13.8	-
17-Aug-2003	14.1	14.100
18-Aug-2003	12.7	13.400
19-Aug-2003	16.5	14.433
20-Aug-2003	15.6	14.933
25-Aug-2003	19.8	19.800
27-Aug-2003	10.7	15.250
28-Aug-2003	14.3	12.500

Table 35.1: A sample moving average

The code to calculate this moving average is:

```
TimeSeries series = new TimeSeries("Series 1", Day.class);
series.add(new Day(11, SerialDate.AUGUST, 2003), 11.2);
series.add(new Day(13, SerialDate.AUGUST, 2003), 13.8);
series.add(new Day(17, SerialDate.AUGUST, 2003), 14.1);
series.add(new Day(18, SerialDate.AUGUST, 2003), 12.7);
series.add(new Day(19, SerialDate.AUGUST, 2003), 16.5);
series.add(new Day(20, SerialDate.AUGUST, 2003), 15.6);
series.add(new Day(25, SerialDate.AUGUST, 2003), 19.8);
series.add(new Day(27, SerialDate.AUGUST, 2003), 10.7);
series.add(new Day(28, SerialDate.AUGUST, 2003), 14.3);

TimeSeries mavg = MovingAverage.createMovingAverage(source,
    "Moving Average", 3, 3);
```

In this example, we have chosen to skip the average calculation for the first three days (11, 12 and 13 August) of the time series (note that there are only two observations in this three day period for the example series). For each of the other dates, an average value is calculated by taking the three days up to and including the particular date. For example, for 19 August, the values for 17, 18 and 19 August are averaged to give a value of 14.433:

[14.1 + 12.7 + 16.5] / 3 = 43.3 / 3 = 14.433

Similarly, the value for 25 August is the average of the values for 23, 24 and 25 August—but in this case no values are available for 23 or 24 August, so only the value from 25 August is used.

35.53.3 Methods

To calculate a moving average for 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 0 or `periodCount - 1`).

To calculate a moving average for each time series in a collection:

```
public static TimeSeriesCollection createMovingAverage(
TimeSeriesCollection source, String suffix, int periodCount, int skip)
```

Returns a new collection containing a moving average time series for each series in the source collection. The names of the moving average series are derived by appending the specified suffix to the source series name.

An alternative means of calculating a moving average is to count back a fixed number of points, irrespective of the "age" of each point:

```
public static TimeSeries createPointMovingAverage(TimeSeries source, String
name, int pointCount)
```

Creates a new series containing moving average values based on the `source` series.

35.53.4 Notes

The `MovingAverageDemo` class in the JFreeChart distribution provides one example of how to use this class.

35.54 MultiIntervalCategoryDataset

35.54.1 Overview

An extension of the `IntervalCategoryDataset` interface that allows multiple intervals for each category.

35.54.2 Notes

The `TaskSeriesCollection` class implements this interface.

35.55 NonGridContourDataset

35.55.1 Overview

A dataset for use with the `ContourPlot` class.

35.56 PieDataset

35.56.1 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.

35.56.2 Methods

This interface adds no methods to those it inherits from the `KeyedValues` and `Dataset` interfaces.

35.56.3 Notes

Some points to note:

- 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.
- you can read a `PieDataset` from a file (in a prespecified XML format) using the `DatasetReader` class.

See Also

`CategoryToPieDataset`, `PiePlot`.

35.57 PowerFunction2D

35.57.1 Overview

A function of the form $y = ax^b$.

35.57.2 Constructor

To construct a new power function:

```
public PowerFunction2D(double a, double b);
```

Creates a new power function with the given coefficients.

35.57.3 Methods

`public double getValue(double x);`
Returns the value of the function for a given input.

35.57.4 Notes

This class implements the `Function2D` interface.

The `RegressionDemo1` application provides an example of this class being used.

See Also
`LineFunction2D`.

35.58 Range

35.58.1 Overview

A class that represents a range of values by recording the lower and upper bounds of the range.

35.58.2 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.

To test whether or not a value falls within the range:

`public boolean contains(double value);`
Returns `true` if *lowerbound* ¡= *value* ¡= *upperbound*, and `false` otherwise.

To combine two ranges:

`public static Range combine(Range range1, Range range2);`
Returns a new range which encompasses both of the specified ranges.

35.58.3 Notes

Some points to note:

- this class is immutable, so instances may be shared;
- the `DateRange` class extends this class to support a date range.

35.59 RangeInfo

35.59.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's range.

35.59.2 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.

35.59.3 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.

35.60 Regression

35.60.1 Overview

This class provides some utility methods for calculating regression co-efficients. Two regression types are supported:

- linear (OLS) regression;
- power regression.

35.60.2 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. The result is an array containing two values, the intercept and the slope.

To calculate a power regression for an array of data values:

> `public static double[] getPowerRegression(double[][] data);`
> Performs a power regression on the data.

35.61 Series

35.61.1 Overview

A useful base class for implementing data series, subclasses include `TimeSeries` and `XYSeries`. This class provides a mechanism for registering *change listeners*, objects that will receive a message (a `SeriesChangeEvent`) every time the series is modified in some way.

35.61.2 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.

35.61.3 Methods

To register a change listener (an object that wishes to receive notification whenever the series is changed):

> `public void addChangeListener(SeriesChangeListener listener);`
> Registers the listener to receive `SeriesChangeEvent` notifications.

To deregister a change listener:

> `public void removeChangeListener(SeriesChangeListener listener);`
> Deregisters the listener.

If you have a lot of changes to make to a series, sometimes it can be a problem that *every* change generates a `SeriesChangeEvent` which is sent to all listeners. You can temporarily disable the event notification using:

> `public void setNotify(boolean notify);`
> Turns the event notification on or off. When you turn this off then on again, a change event is sent immediately.

See Also

`AbstractSeriesDataset`, `TimeSeries`, `XYSeries`.

35.62 SeriesChangeEvent

35.62.1 Overview

An event class that is passed to a `SeriesChangeListener` to notify it concerning a change to a `Series`.

35.63 SeriesChangeListener

35.63.1 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`).

35.63.2 Methods

This interface defines a single method:

```
public void seriesChanged(SeriesChangeEvent event);
```

Receives notification when a series changes.

35.63.3 Notes

The `AbstractSeriesDataset` class implements this interface—it will generate a `DatasetChangeEvent` every time it receives notification of a `SeriesChangeEvent`.

35.64 SeriesDataset

35.64.1 Overview

A base interface that defines a dataset containing zero, one or many data series.

35.64.2 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).

35.64.3 Notes

This interface is extended by `CategoryDataset` and `XYDataset`.

35.65 SeriesException

35.65.1 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`.

35.66 SignalsDataset

35.66.1 Overview

Not yet documented.

35.67 SubseriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

35.68 TableXYDataset

35.68.1 Overview

This interface is an extension of the `XYDataset` interface. By implementing this interface, a dataset is declaring that all series share a common set of x-values—this is required by renderers that "stack" values (for example, the `StackedXYAreaRenderer`).

35.69 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

35.70 Value

35.70.1 Overview

An interface for accessing a single value (`Number` object). By way of an example, the `ValueDataset` interface extends this interface, and is used by the `ThermometerPlot` class.

35.70.2 Methods

The interface defines a single method for accessing the value:

```
public Number getValue();
```

Returns the value (possibly `null`).

35.70.3 Notes

Some notes:

- the `KeyedValue` interface extends this interface.
- the `DefaultKeyedValue` class provides one implementation of this interface.

35.71 ValueDataset

35.71.1 Overview

A *value dataset* stores a single value (`Number` object).

35.71.2 Methods

This interface extends the `Value` and `Dataset` interfaces, and adds no new methods.

35.71.3 Notes

This dataset is used by the `ThermometerPlot` class.

35.72 Values

35.72.1 Overview

An interface for accessing a collection of values.

35.72.2 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`).

35.72.3 Notes

Some notes:

- the `KeyedValues` interface extends this interface.
- the `DefaultKeyedValues` class provides one implementation of this interface.

35.73 Values2D

35.73.1 Overview

An interface for accessing a table of values.

35.73.2 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.

35.73.3 Notes

Some points to note:

- the `KeyedValues2D` interface extends this interface.
- the `DefaultKeyedValues2D` class provides one implementation of this interface.

35.74 WindDataset

35.74.1 Overview

A *wind dataset* provides wind direction and intensity values observed at various points in time.

35.74.2 Notes

The `WindChartDemo` application, included in the JFreeChart distribution, provides an example.

35.75 XisSymbolic

35.75.1 Overview

Not yet documented.

35.76 XYBarDataset

35.76.1 Overview

A dataset wrapper class that can convert any `XYDataset` into an `IntervalXYDataset`.

35.76.2 Constructor

To create a new dataset wrapper:

> `public XYBarDataset(XYDataset underlying, double barWidth);`
> Creates a wrapper for the underlying dataset, effectively converting it into an `IntervalXYDataset`.

35.77 XYDataItem

35.77.1 Overview

This class represents a pair *(x, y)* of `Number` objects. The x-value should always be defined, but the y-value can be set to `null` to represent a missing or unknown value.

35.77.2 Notes

Some notes:

- this class implements the `Comparable` interface, and implements ordering by x-values.
- this class parallels the `TimeSeriesDataItem` class.

35.78 XYDataset

35.78.1 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`, `XYZDataset` and `TableXYDataset`.

35.78.2 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`).

35.78.3 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`.

35.79 XYSeries

35.79.1 Overview

A series of *(x, y)* data items (extends `Series`). Each item is represented by an instance of `XYDataItem` and stored in a list (sorted in ascending order of x-values, by default).

`XYSeries` will allow duplicate x-values, unless a flag is set in the constructor to prevent duplicates.

35.79.2 Constructors

To construct a series:

`public XYSeries(String name);`
Creates a new series (initially empty) with the specified name. By default, the data items will be sorted in ascending order of x-values, and duplicate x-values will be allowed.

To construct a series with control over sorting and whether or not duplicate x-values are permitted:

```
public XYSeries(String name,
boolean autoSort, boolean allowDuplicateXValues);
```
Creates a new series (initially empty) with the specified name. Flags are set that determine whether the data items are sorted by x-value, and where duplicate x-values will be allowed or disallowed, as specified.

35.79.3 Methods

To find out how many items are contained in a series:

> `public int getItemCount();`
> Returns the number of items in the series.

You can obtain a list of the items in the dataset:

> `public List getItems();`
> Returns an unmodifiable list of the items in the series. Note that the list is unmodifiable, but you can still change the y-values for the individual data items in the list—this is not the recommended way to change data in the series, because no notification of the change occurs.

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.

35.79.4 Notes

Some points to note:

- 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.

35.80 XYSeriesCollection

35.80.1 Overview

A collection of `XYSeries` objects. This class implements both the `XYDataset` and `IntervalXYDataset` interfaces, so can be used as the dataset for a wide range of charts.

35.80.2 Constructors

To construct a series collection:

> `public XYSeriesCollection();`
> Creates a new empty series collection.

35.80.3 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.

35.80.4 Using as an `IntervalXYDataset`

This class implements the `IntervalXYDataset` interface, which means you can (for example) use the collection as a dataset to create a bar chart (using the `XYPlot` and `XYBarRenderer` classes). The underlying data items are just points, so it is necessary to "manufacture" an x-interval for each item. The width of this interval defaults to `1.0`, but can be specified with the following method:

> `public void setIntervalWidth(double width);`
> Sets the width of the x-interval and sends a `DatasetChangeEvent` to all registered listeners.

Given a data item at *(2.0, 3.75)*, the default x-interval will be extend from `1.5` to `2.5` (that is, an interval of width `1.0` centered about the x-value of `2.0`). You might want to change where the interval falls about the actual x-value—you can use the following method:

> `public void setIntervalPositionFactor(double factor);`
> Sets the interval position factor, a value between 0.0 and 1.0 (the default is 0.5, which centers the interval about the x-value).

35.81 XYZDataset

35.81.1 Overview

An interface that defines a collection of data items in the form of (x, y, z) values. This is a natural extension of the `XYDataset` interface.

35.81.2 Notes

JFreeChart doesn't have support for three dimensional charts yet, but this interface still finds a use in the `XYBubbleRenderer` class.

35.82 YisSymbolic

35.82.1 Overview

To be documented.

Chapter 36

Package: org.jfree.data.gantt

36.1 Introduction

This package contains classes used to represent the dataset for a Gantt chart.

36.2 GanttCategoryDataset

36.2.1 Overview

An extension of the IntervalCategoryDataset interface that is intended for creating Gantt charts.

36.2.2 Methods

This interface adds a range of methods in addition to those it inherits from the IntervalCategoryDataset interface. These are aimed at supporting subtasks within tasks, and providing information about the "percentage complete" for individual tasks.

To get the number of subtasks for a given task:

> `public int getSubIntervalCount(int row, int column);`
> Returns the number of subtasks defined for the specified item (possibly `0`).

> `public int getSubIntervalCount(Comparable rowKey, Comparable columnKey);`
> Returns the number of subtasks defined for the specified item (possibly `0`).

To get the start value (time in milliseconds) for a specific subtask:

> `public Number getStartValue(int row, int column, int subinterval);`
> Returns the start value for a subtask.

> `public Number getStartValue(Comparable rowKey, Comparable columnKey, int subinterval);`
> Returns the start value for a subtask.

To get the end value (time in milliseconds) for a specific subtask:

> `public Number getEndValue(int row, int column, int subinterval);`
> Returns the end value for a subtask.

> `public Number getEndValue(Comparable rowKey, Comparable columnKey, int subinterval);`
> Returns the end value for a subtask.

To get the percentage complete for a given task:

> `public Number getPercentComplete(int row, int column);`
> Returns the percentage complete for the specified task. This method can return `null` if the value is unknown.

> `public Number getPercentComplete(Comparable rowKey, Comparable columnKey);`
> Returns the percentage complete for the specified task. This method can return `null` if the value is unknown.

To get the percentage complete for a subtask:

> `public Number getPercentComplete(int row, int column, int subinterval);`
> Returns the percentage complete for the specified subtask. This method can return `null` if the value is unknown.

> `public Number getPercentComplete(Comparable rowKey, Comparable columnKey, int subinterval);`
> Returns the percentage complete for the specified subtask. This method can return `null` if the value is unknown.

36.2.3 Notes

The `GanttRenderer` class expects to find a dataset of this type.

36.3 Task

36.3.1 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`.

36.4 TaskSeries

36.4.1 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*.

36.5 TaskSeriesCollection

36.5.1 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*.

Chapter 37

Package: org.jfree.data.statistics

37.1 Introduction

This package contains interfaces and classes for representing statistical datasets.

37.2 BoxAndWhiskerCalculator

37.2.1 Overview

A utility class for calculating the statistics required for a box-and-whisker plot.

37.2.2 Methods

To calculate box-and-whisker statistics for a list of values:

> `public static BoxAndWhiskerItem calculateBoxAndWhiskerStatistics(List values);`
> Calculates a set of statistics (mean, median, quartiles Q1 and Q3, plus outliers) for a list of `Number` objects.

To calculate the mean of a list of values:

> `public static double calculateMean(List values)`
> Returns the mean of a list of numbers. Items in the list that are not instances of the `Number` class are ignored. Likewise, `null` items are ignored.

To calculate the median of a list of values:

> `public static double calculateMedian(List values);`
> Returns the median of a list of values. This method REQUIRES the list of values to be in ascending order.

To calculate the first quartile value:

> `public static double calculateQ1(List values);`
> Returns the first quartile boundary for a list of values. This method REQUIRES the list of values to be in ascending order.

To calculate the third quartile value:

> `public static double calculateQ3(List values);`
> Returns the first quartile boundary for a list of values. This method REQUIRES the list of values to be in ascending order.

37.3 BoxAndWhiskerCategoryDataset

37.3.1 Overview

An interface that extends the `CategoryDataset` interface and returns the values required for a box-and-whisker chart. The dataset represents a two-dimensional table, where each cell in the table contains a complete set of statistics for one box-and-whisker item (a mean, median, quartile boundary values Q1 and Q3, plus information about outliers and farouts).

The `DefaultBoxAndWhiskerCategoryDataset` provides one implementation of this interface.

37.3.2 Methods

The interface provides a range of methods for reading the values from the dataset. No update methods are provided, since not every dataset implementation needs to be writeable.

To get the mean for one item in the dataset:

> `public Number getMeanValue(int row, int column);`
> Returns the mean value for an item.

> `public Number getMeanValue(Comparable rowKey, Comparable columnKey);`
> Returns the mean value for an item.

To get the median value for one item in the dataset:

> `public Number getMedianValue(int row, int column);`
> Returns the median value for an item.

> `public Number getMedianValue(Comparable rowKey, Comparable columnKey);`
> Returns the median value for an item.

To get the first quartile boundary value:

> `public Number getQ1Value(int row, int column);`
> Returns the first quartile boundary value.

> `public Number getQ1Value(Comparable rowKey, Comparable columnKey);`
> Returns the first quartile boundary value.

To get the third quartile boundary value:

> `public Number getQ3Value(int row, int column);`
> Returns the third quartile boundary value.

`public Number getQ3Value(Comparable rowKey, Comparable columnKey);`
Returns the third quartile boundary value.

To get the minimum regular value (everything lower than this is either an outlier or a farout):

`public Number getMinRegularValue(int row, int column);`
Returns the lowest regular value.

`public Number getMinRegularValue(Comparable rowKey, Comparable columnKey);`
Returns the lowest regular value.

To get the maximum regular value (everything higher than this is either an outlier or a farout):

`public Number getMaxRegularValue(int row, int column);`
Returns the highest regular value.

`public Number getMaxRegularValue(Comparable rowKey, Comparable columnKey);`
Returns the highest regular value.

To get the minimum outlier (everything lower than this is a farout value):

`public Number getMinOutlier(int row, int column);`
Returns the lowest outlier.

`public Number getMinOutlier(Comparable rowKey, Comparable columnKey);`
Returns the lowest outlier.

To get the maximum outlier (everything higher than this is a farout value):

`public Number getMaxOutlier(int row, int column);`
Returns the highest outlier.

`public Number getMaxOutlier(Comparable rowKey, Comparable columnKey);`
Returns the highest outlier.

To get a list of the outlier (and farout) values for an item in the dataset:

`public List getOutliers(int row, int column);`
Returns a list of the outlier (and farout) values.

`public List getOutliers(Comparable rowKey, Comparable columnKey);`
Returns a list of the outlier (and farout) values.

37.4 BoxAndWhiskerItem

37.4.1 Overview

A small class that holds the statistics and values required for a box-and-whisker item:

- a mean;

- a median;
- a first quartile boundary value;
- a third quartile boundary value;
- a minimum regular value;
- a maximum regular value;
- a minimum outlier;
- a maximum outlier;
- a list of outlier values;

This class is immutable.

37.4.2 Notes

The `BoxAndWhiskerCalculator` class returns instances of this class from one of its methods.

37.5 BoxAndWhiskerXYDataset

37.5.1 Overview

An interface that is used to obtain data for a box-and-whisker plot using the `XYPlot` class. This interface extends `XYDataset`.

The `DefaultBoxAndWhiskerXYDataset` class provides one implementation of this interface.

37.5.2 Methods

To get the mean value for an item:

> `public Number getMeanValue(int series, int item);`
> Returns the mean value.

To get the median value for an item:

> `public Number getMedianValue(int series, int item);`
> Returns the median value.

To get the first quartile boundary value:

> `public Number getQ1Value(int series, int item);`
> Returns the first quartile boundary value.

To get the third quartile boundary value:

> `public Number getQ3Value(int series, int item);`
> Returns the third quartile boundary value.

To get the minimum regular value:

`public Number getMinRegularValue(int series, int item);`
Returns the minimum regular value. Anything lower than this is either an outlier or a farout value.

To get the maximum regular value:

`public Number getMaxRegularValue(int series, int item);`
Returns the maximum regular value. Anything higher than this is either an outlier or a farout value.

To get the minimum outlier:

`public Number getMinOutlier(int series, int item);`
Returns the minimum outlier. Anything lower than this is a farout value.

To get the maximum outlier:

`public Number getMaxOutlier(int series, int item);`
Returns the maximum outlier. Anything higher than this is a farout value.

To get a list of the outlier values:

`public List getOutliers(int series, int item);`
Returns a list of the outlier (and farout) values for this item.

To get the outlier coefficient:

`public double getOutlierCoefficient();`
Returns the outlier coefficient (this is probably redundant).

To get the farout coefficient:

`public double getFaroutCoefficient();`
Returns the farout coefficient (this is probably redundant).

37.6 DefaultBoxAndWhiskerCategoryDataset

37.6.1 Overview

A basic implementation of the `BoxAndWhiskerCategoryDataset` interface.

37.6.2 Notes

The `BoxAndWhiskerDemo` (included in the JFreeChart distribution) provides an example of this class being used.

37.7 DefaultBoxAndWhiskerXYDataset

37.7.1 Overview

A basic implementation of the `BoxAndWhiskerXYDataset` interface.

37.7.2 Notes

The `XYBoxAndWhiskerDemo` (included in the JFreeChart distribution) provides an example of this class being used.

37.8 DefaultStatisticalCategoryDataset

37.8.1 Overview

A default implementation of the `StatisticalCategoryDataset` interface.

37.9 HistogramBin

37.9.1 Overview

This class is used to represent a bin for the `HistogramDataset` class.

37.10 HistogramDataset

37.10.1 Overview

A dataset that can be used with the `XYPlot` class to display a histogram.

37.10.2 Constructors

The default constructor creates an empty dataset:

> `public HistogramDataset();`
> Creates an empty dataset with a type of `HistogramType.FREQUENCY`.

37.10.3 Methods

To set the type of histogram:

> `public void setType(HistogramType type);`
> Sets the histogram type and sends a `DatasetChangeEvent` to all registered listeners.

To add raw data to the dataset, allowing the bin range to be determined automatically to fit the data:

> `public void addSeries(String name, double[] values, int bins);`
> Creates a series within the dataset that summarises the values supplied by allocating them to the specified number of bins. The bin size is calculated to cover the range of values in the array.

To add raw data to the dataset, using a specified bin range:

> `public void addSeries(String name, double[] values, int bins, double minimum, double maximum);`
> Creates a series within the dataset the summarises the values supplied by allocating them to bins. The bin size is calculated so that the specified number of bins covers the range *(minimum, maximum)*.

For both of the above methods, values that fall on a bin boundary will be allocated to the *lower* bin (except in the case of the *minimum* value which is assigned to the first bin).

37.10.4 Notes

Some points to note:

- the dataset is `Cloneable` and `Serializable`;
- a demo (`HistogramDemo.java`) is included in the JFreeChart distribution, in the `src/org/jfree/chart/demo` directory.

37.11 HistogramType

37.11.1 Overview

An enumeration of the possible histogram types:

- `FREQUENCY` - a *frequency histogram* shows the number of data items allocated to each bin;
- `RELATIVE_FREQUENCY` - a *relative frequency histogram* shows the number of data items allocated to each bin as a fraction of the total number of items;
- `SCALE_AREA_TO_1` - similar to a relative frequency histogram, except that the values are scaled so that the overall area represented by the bars is equal to 1.

37.11.2 Usage

These values are normally used in the `getType()` and `setType()` methods of the `HistogramDataset` class.

37.12 StatisticalCategoryDataset

37.12.1 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.

37.12.2 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.

37.12.3 Notes

The `DefaultStatisticalCategoryDataset` class implements this interface.

37.13 Statistics

37.13.1 Overview

Provides some static utility methods for calculating statistics.

37.13.2 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.

37.13.3 Notes

This class was written by Matthew Wright.

Chapter 38

Package: org.jfree.data.time

38.1 Introduction

This package contains interfaces and classes that are used to represent *time-based* data.

The TimeSeriesCollection class is perhaps the most important class in this package. It is used to store one or more TimeSeries objects, and provides an implementation of the XYDataset interface. This allows it to be used as the dataset for an XYPlot).

The TimePeriodValuesCollection class performs a similar role, but allows more general (less regular) time periods to be used.

38.2 Day

38.2.1 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.

38.2.2 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.

38.2.3 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.

38.2.4 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.

38.2.5 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.

38.3 FixedMillisecond

38.3.1 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`.

38.3.2 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.

38.3.3 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.

38.3.4 Notes

Some 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.

38.4 Hour

38.4.1 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`.

38.4.2 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.

38.4.3 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.

38.4.4 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.

38.4.5 Notes

The `Hour` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

38.5 Millisecond

38.5.1 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`.

38.5.2 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.

38.5.3 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.

38.5.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

38.6 Minute

38.6.1 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.

38.6.2 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.

38.6.3 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.

38.6.4 Notes

The `Minute` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

38.7 Month

38.7.1 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`.

38.7.2 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.

38.7.3 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.

38.7.4 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.

38.8 Quarter

38.8.1 Overview

A calendar quarter—this class extends `RegularTimePeriod`.

38.8.2 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);
```

38.8.3 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.

38.8.4 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.

38.8.5 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.

38.9 RegularTimePeriod

38.9.1 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`.

38.9.2 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.[1]

38.9.3 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.

[1] 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.

38.9.4 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).
>
> `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).

38.9.5 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`.

38.10 Second

38.10.1 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`.

38.10.2 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.

38.10.3 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.

38.10.4 Notes

The `Second` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

38.11 SimpleTimePeriod

38.11.1 Overview

A simple implementation of the `TimePeriod` interface.

38.11.2 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.

38.12 TimePeriod

38.12.1 Overview

A period of time defined by two `java.util.Date` instances representing the start and end of the time period.

38.12.2 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.

38.12.3 Notes

This interface is implemented by:

- the `SimpleTimePeriod` class;
- the `RegularTimePeriod` base class and all its subclasses.

38.13 TimePeriodAnchor

38.13.1 Overview

An enumeration of the three possible *time period anchor positions*:

- `START` - the start of the time period;
- `MIDDLE` - the middle of the time period;
- `END` - the end of the time period.

These are used by the `TimeSeriesCollection` and `TimePeriodValuesCollection` classes to determine how x-values are derived from the underlying time periods when these classes are used as `XYDataset` instances.

38.14 TimePeriodFormatException

38.14.1 Overview

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

38.15 TimePeriodValue

38.15.1 Overview

An object that represents a time period with an associated value, used to represent each item in a `TimePeriodValues` collection.

38.15.2 Constructors

To create a new `TimePeriodValue` object:

> `public TimePeriodValue(TimePeriod period, Number value);`
> Creates a new data item that associates a value (`null` permitted) with a period.

For convenience, you can also use the following constructor:

> `public TimePeriodValue(TimePeriod period, double value);`
> Creates a new data item that associates a value with a period.

38.15.3 Methods

There are methods for accessing the `period` and `value` attributes. You can update the value but not the period (this allows other classes to maintain a collection of `TimePeriodValue` objects in some order that is based on the `period`, without the risk of that order being compromised by a change to a particular item).

38.16 TimePeriodValues

38.16.1 Overview

A collection of `TimePeriodValue` objects. The objects are maintained in the order they are added. This class is used to represent one data series in a `TimePeriodValuesCollection`.

38.17 TimePeriodValuesCollection

38.17.1 Overview

A collection of `TimePeriodValues` objects.

38.17.2 Usage

The `TimePeriodValuesDemo` application, included in the JFreeChart distribution, provides an example of how to use this class.

38.17.3 Constructors

To create a new, empty collection:

> `public TimePeriodValuesCollection();`
> Creates a new empty collection. After creation, you can add `TimePeriodValues` objects using the `addSeries(...)` method.

38.17.4 Methods

To add a new series to the collection:

> `public void addSeries(TimePeriodValues series);`
> Adds a series to the collection. A `DatasetChangeEvent` is sent to all registered listeners.

38.17.5 Notes

This class implements the `DomainInfo` interface.

38.18 TimeSeries

38.18.1 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`.

The values are represented by the `Number` class. The value `null` can be used to indicate missing or unknown values.

38.18.2 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.

The TimeSeriesDemo class provides an example of how to create a dataset for a chart using this class.

38.18.3 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.

38.18.4 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".
maximum-item-count	The maximum number of items that the series will record. Once this limit is reached, the oldest observation is dropped whenever a new observation is added.
history-count	The number of time periods defining a "window" for the data. Starting with the latest observation, the window extends back for this number of time periods. Any data older than the window is discarded.

38.18.5 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 getDataItem(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 getDataItem(linkRegularTimePeriod 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.

38.18.6 Notes

You can calculate the moving average of a time series using the `MovingAverage` utility class.

See Also

`TimePeriod`, `TimeSeriesCollection`.

38.19 TimeSeriesCollection

38.19.1 Overview

A collection of `TimeSeries` objects. A key feature of this class is that it implements the `XYDataset` interface, which means that you can use it to generate time series charts easily. Further, it implements the `IntervalXYDataset` interface, which is used by some specialised chart renderers.

The *xPosition* and *domainIsPointsInTime* attributes control aspects of the behaviour of this class when it is being used as a dataset—see section 38.19.4 for details.

38.19.2 Usage

The `TimeSeriesDemo` application, included in the JFreeChart distribution, provides an example of how to use this class.

38.19.3 Constructors

To create an *empty* time series collection:

> `public TimeSeriesCollection();`
> Creates a new, empty collection.

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.

38.19.4 Attributes

When this class is used an an `XYDataset`, the *xPosition* attribute is used to determine how each x-value is derived from the underlying time period for a data item. You can choose to return the *start*, *middle* (the default) or *end* of the time period as the x-value.

The *domainIsPointsInTime* flag controls the treatment of time periods in the collection when the overall range of values is being calculated. There are two possibilities:

- consider each time period as a single point, which is the case when the collection is being used as an `XYDataset`;
- consider each time period as a range of values, which is the case when the collection is being used as an `IntervalXYDataset`.

If the *domainIsPointsInTime* flag is set to `TRUE` (the default), the former treatment is applied, and if it is set to `FALSE` the latter treatment is applied.

38.19.5 Methods

To find out how many `TimeSeries` 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.

To alter the way that x-values are derived from the underlying time period:

> `public void setXPosition(TimePeriodAnchor anchor);`
> Sets the position (`START`, `MIDDLE`, or `END`) within each time period that is used as the x-value for a data item.

The `DomainInfo` interface requires the following method, which returns the overall range of x-values contained in the collection:

> `public Range getDomainRange();`
> Returns the overall range of x-values contained in the collection. The result is affected by the current setting of the *domainIsPointsInTime* attribute—see section 38.19.4 for details.

38.19.6 Notes

Points to note:

- this class extends `AbstractSeriesDataset` to provide some of the basic series information.
- this class implements the `XYDataset` and `IntervalXYDataset` interfaces.

38.20 TimeSeriesDataItem

38.20.1 Overview

This class associates a `Number` with a `RegularTimePeriod`, and is used by the `TimeSeries` class to record individual data items.

38.20.2 Usage

You won't normally use this class directly. The `TimeSeries` class will create instances as required.

38.20.3 Notes

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.
- instances of this class can be easily cloned.

38.21 Week

38.21.1 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.

38.21.2 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.

38.21.3 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 RegularTimePeriod previous();`
> Returns the previous week, or null if the lower limit of the range is reached.

> `public RegularTimePeriod 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.

38.21.4 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 `RegularTimePeriod` subclasses.

See Also:
`Year`.

38.22 Year

38.22.1 Overview

A class that represents a calendar year (for example, "2003"). This class extends `RegularTimePeriod`.

38.22.2 Usage

A typical use for this class is for creating `TimeSeries` objects for *annual data*. For example:

```
TimeSeries t1 = new TimeSeries("Series 1", "Year", "Value", Year.class);
t1.add(new Year(1990), new Double(50.1));
t1.add(new Year(1991), new Double(12.3));
t1.add(new Year(1992), new Double(23.9));
t1.add(new Year(1993), new Double(83.4));
t1.add(new Year(1994), new Double(-34.7));
t1.add(new Year(1995), new Double(76.5));
t1.add(new Year(1996), new Double(10.0));
t1.add(new Year(1997), new Double(-14.7));
t1.add(new Year(1998), new Double(43.9));
t1.add(new Year(1999), new Double(49.6));
t1.add(new Year(2000), new Double(37.2));
t1.add(new Year(2001), new Double(17.1));
```

38.22.3 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.

38.22.4 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.

38.22.5 Notes

Some points to note:

- 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.

Chapter 39

Package: org.jfree.data.xml

39.1 Introduction

This package contains interfaces and classes that provide basic support for reading datasets from XML files. In the current release, there is support for `PieDataset` and `CategoryDataset`. It is intended that other dataset types will be supported in the future.

39.2 Usage

In normal usage, you will access the facilities provided by this package via methods in the `DatasetReader` class. The following examples are provided in the JFreeChart distribution:

- `XMLBarChartDemo.java`
- `XMLPieChartDemo.java`

You will find these demos in the `src/org/jfree/chart/demo` directory.

39.3 CategoryDatasetHandler

39.3.1 Overview

A SAX handler that creates a `CategoryDataset` by processing the elements in an XML document.

39.3.2 Usage

In most cases, you won't need to use this class directly. Instead, use the `DatasetReader` class. For an example, see the `XMLBarChartDemo` included in the JFreeChart distribution.

39.3.3 XML Format

The format supported by the handler is illustrated by the following example:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Sample data for JFreeChart. -->

<CategoryDataset>

  <Series name = "Series 1">
    <Item>
      <Key>Category 1</Key>
      <Value>15.4</Value>
    </Item>
    <Item>
      <Key>Category 2</Key>
      <Value>12.7</Value>
    </Item>
    <Item>
      <Key>Category 3</Key>
      <Value>5.7</Value>
    </Item>
    <Item>
      <Key>Category 4</Key>
      <Value>9.1</Value>
    </Item>
  </Series>

  <Series name = "Series 2">
    <Item>
      <Key>Category 1</Key>
      <Value>45.4</Value>
    </Item>
    <Item>
      <Key>Category 2</Key>
      <Value>73.7</Value>
    </Item>
    <Item>
      <Key>Category 3</Key>
      <Value>23.7</Value>
    </Item>
    <Item>
      <Key>Category 4</Key>
      <Value>19.4</Value>
    </Item>
  </Series>

</CategoryDataset>
```

The `<CategoryDataset>` element can contain any number of `<Series>` elements, and each `<Series>` element can contain any number of `<Item>` elements.

39.3.4 Notes

This class delegates work to the `CategorySeriesHandler` class.

39.4 CategorySeriesHandler

39.4.1 Overview

A SAX handler that reads a `<Series>` sub-element within a category dataset XML file. Work is delegated to this class by the `CategoryDatasetHandler` class.

39.5 DatasetReader

39.5.1 Overview

This class contains utility methods for reading datasets from XML files. In the current release, support is included for `PieDataset` and `CategoryDataset`.

39.5.2 Usage

Two applications (`XMLPieChartDemo` and `XMLBarChartDemo`) that demonstrate how to use this class are included in the JFreeChart distribution.

39.6 DatasetTags

39.6.1 Overview

An interface that defines constants for the literal text used in the element tags within the XML documents.

Attribute:	**Value:**
`PIEDATASET_TAG`	`PieDataset`
`CATEGORYDATASET_TAG`	`CategoryDataset`
`SERIES_TAG`	`Series`
`ITEM_TAG`	`Item`
`KEY_TAG`	`Key`
`VALUE_TAG`	`Value`

Table 39.1: Attributes for the `DatasetTags` interface

39.7 ItemHandler

39.7.1 Overview

A SAX handler that reads a *key/value* pair.

39.7.2 Usage

You should not need to use this class directly. Work is delegated to this handler by the `PieDatasetHandler` class.

39.7.3 Notes

This class delegates some work to the `KeyHandler` class.

39.8 KeyHandler

39.8.1 Overview

A SAX handler that reads a *key* element from an XML file.

39.8.2 Usage

You should not need to use this class directly. Work is delegated to this class by the ItemHandler class.

39.8.3 Notes

A key can be any instance of Comparable, but the handler always uses the String class to represent keys.

39.9 PieDatasetHandler

39.9.1 Overview

A SAX handler for reading a PieDataset from an XML file.

39.9.2 Usage

In most cases, you won't need to use this class directly. Instead, use the DatasetReader class. For an example, see the XMLPieChartDemo application included in the JFreeChart distribution.

39.9.3 XML Format

The format supported by the handler is illustrated by the following example:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- A sample pie dataset for JFreeChart. -->

<PieDataset>
  <Item>
    <Key>Java</Key>
    <Value>15.4</Value>
  </Item>
  <Item>
    <Key>C++</Key>
    <Value>12.7</Value>
  </Item>
  <Item>
    <Key>PHP</Key>
    <Value>5.7</Value>
  </Item>
  <Item>
    <Key>Python</Key>
    <Value>9.1</Value>
  </Item>
</PieDataset>
```

The <PieDataset> element can contain any number of <Item> elements.

39.9.4 Notes

This class delegates some work to the ItemHandler class.

39.10 RootHandler

39.10.1 Overview

The base handler class that provides support for a "sub-handler stack". While processing an XML element, a handler can push a sub-handler onto the stack and delegate work to it (usually the processing of a sub-element). When the sub-handler is finished its work, it gets popped from the stack, and the original handler resumes control. In this way, nested elements within the XML file can be processed by different classes.

39.11 ValueHandler

39.11.1 Overview

A SAX handler that processes numerical values.

Appendix A

JCommon

A.1 Introduction

JFreeChart makes use of classes in the JCommon class library. The JCommon runtime jar file is included in the JFreeChart distribution. If you require the source code and/or documentation, you can download these from:

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

Selected JCommon classes are documented here because they are used extensively within JFreeChart.

A.2 PublicCloneable

A.2.1 Overview

An interface for objects with a `clone()` method. This is used in JFreeChart to "look behind" an interface to see if the class implementing the interface can be cloned.

A.2.2 Methods

This interface declares a single method:

```
public Object clone() throws CloneNotSupportedException;
```

Creates a clone of the object.

A.3 RectangleAnchor

A.3.1 Overview

This class defines an enumeration of nine common anchor points within a rectangle. These points include the four corners of the rectangle, the four mid-points of each rectangle edge, and the center point:

ID:	Description:
RectangleAnchor.TOP	The midpoint of the rectangle's top edge.
RectangleAnchor.BOTTOM	The midpoint of the rectangle's bottom edge.
RectangleAnchor.LEFT	The midpoint of the rectangle's left edge.
RectangleAnchor.RIGHT	The midpoint of the rectangle's right edge.
RectangleAnchor.TOP_LEFT	The top-left corner of the rectangle.
RectangleAnchor.TOP_RIGHT	The top-right corner of the rectangle.
RectangleAnchor.BOTTOM_LEFT	The bottom-left corner of the rectangle.
RectangleAnchor.BOTTOM_RIGHT	The bottom-right corner of the rectangle.
RectangleAnchor.CENTER	The center of the rectangle.

Table A.1: Constants defined by `RectangleAnchor`

A.4 RectangleEdge

A.4.1 Overview

This class defines an enumeration of the four edges of a rectangle. It is used to specify the location of objects (for example, axes in a plot) relative to a rectangle:

ID:	Description:
RectangleEdge.TOP	The top edge.
RectangleEdge.BOTTOM	The bottom edge.
RectangleEdge.LEFT	The left edge.
RectangleEdge.RIGHT	The right edge.

Table A.2: Constants defined by `RectangleEdge`

A.5 Spacer

A.5.1 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).

A.5.2 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.

A.5.3 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.

A.5.4 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.

A.6 TextAnchor

A.6.1 Overview

This class defines an enumeration of the anchor points relative to the bounds of a text string (see table A.3). It is used to specify an anchor point for text alignment and rotation.

ID:	**Description:**
`TextAnchor.TOP_LEFT`	The top left corner.
`TextAnchor.TOP_CENTER`	The center point on the top edge.
`TextAnchor.TOP_RIGHT`	The top right corner.
`TextAnchor.CENTER_LEFT`	The center point on the left edge.
`TextAnchor.CENTER`	The center point of the text.
`TextAnchor.CENTER_RIGHT`	The center point on the right edge.
`TextAnchor.HALF_ASCENT_LEFT`	The half ascent point on the left edge.
`TextAnchor.HALF_ASCENT_CENTER`	The center point along the half ascent line.
`TextAnchor.HALF_ASCENT_RIGHT`	The half ascent point on the right edge.
`TextAnchor.BASELINE_LEFT`	The baseline point on the left edge.
`TextAnchor.BASELINE_CENTER`	The center point along the half ascent line.
`TextAnchor.BASELINE_RIGHT`	The baseline point on the right edge.
`TextAnchor.BOTTOM_LEFT`	The bottom left corner.
`TextAnchor.BOTTOM_CENTER`	The center point on the bottom edge.
`TextAnchor.BOTTOM_RIGHT`	The bottom right corner.

Table A.3: Constants defined by `TextAnchor`

Appendix B

The GNU Lesser General Public License

B.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public License (LGPL). The full text of this license is reproduced in this appendix. You should read and understand this license 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 B.3 first).

B.2 The License

The following license has been used for the distribution of the JFreeChart class library:

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[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!

B.3 Frequently Asked Questions

B.3.1 Introduction

Some of the most frequently asked questions about JFreeChart concern the license. I've published this FAQ to help developers understand my choice of license 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.

B.3.2 Questions and Answers

1. "Can I incorporate JFreeChart into a proprietary (closed-source) application?"

Yes, the GNU Lesser General Public License (LGPL) is specifically designed to allow this.

2. "Do I have to pay a license 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 license, which (for most developers) is not very difficult.

If you want to make a financial contribution to the JFreeChart project, you can buy a copy of the JFreeChart Developer Guide from Object Refinery Limited. 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 license 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 license (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 license;

- 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 2000-2004 by Object Refinery 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 License (GPL). It is a much simpler license, 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 license 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 license. 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 license?"

The license 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 license, such as an "Apache-style" license?"

Yes, a range of licenses was considered for JFreeChart, but now that the choice has been made there are no plans to change the license in the future.

A publication by Bruce Perens was especially helpful in comparing the available licenses:

`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 license, 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 licenses (for example the Apache Software License) allow open source software to be packaged and redistributed without source code. These licenses 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.

Index