The JFreeChart Class Library

Version 0.9.1

REFERENCE DOCUMENTATION

Written by David Gilbert

June 14, 2002

Contents

1 Introduction

1.1 What is JFreeChart?

JFreeChart is a free[1] Java class library for generating charts.

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

JFreeChart has the following features:

- interactive zooming;
- events;
- tooltips;
- data is accessible from any implementation of the defined interfaces;
- export to JPEG, PNG, SVG, PDF and any other format with a `Graphics2D` implementation;
- works in applications, servlets and applets;
- complete source code available under the terms of the GNU Lesser General Public License (LGPL);

JFreeChart can be downloaded from:

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

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

1.2 This Document

This document has been written for version 0.9.1 of JFreeChart.

Two versions of the document are available:

- a free version can be downloaded from the JFreeChart web page, and includes the chapters up to and including the instructions for installing JFreeChart.

[1] *Free* under the terms of the GNU Lesser General Public License. See Appendix A for details.

- a premium version can be purchased from the JFreeChart web page and includes additional tutorial chapters and reference documentation for the JFreeChart classes. Proceeds from the sale of this document are used to sponsor on-going development of JFreeChart.

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 their contributions: Andrzej Porebski, Bill Kelemen, David Berry, Matthew Wright, David Li, Sylvain Vieujot, Serge V. Grachov, Jonathan Nash, Hans-Jurgen Greiner, Joao Guilherme Del Valle, Mark Watson, Søren Caspersen, Laurence Vanhelsuwé, Martin Cordova, Wolfgang Irler, Craig MacFarlane, Michael Duffy, Bryan Scott, Hari, Anthony Boulestreau, Thomas Meier, Sam (oldman), Jeremy Bowman, Jean-Luc Schwab, Roger Studner, Andreas Schneider, Eric Thomas, Jon Iles, Tin Luu and Krzysztof Paz.

1.4 Comments and Suggestions

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

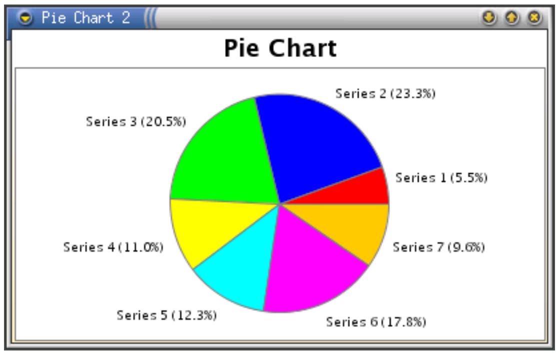
2 Sample Charts

2.1 Introduction

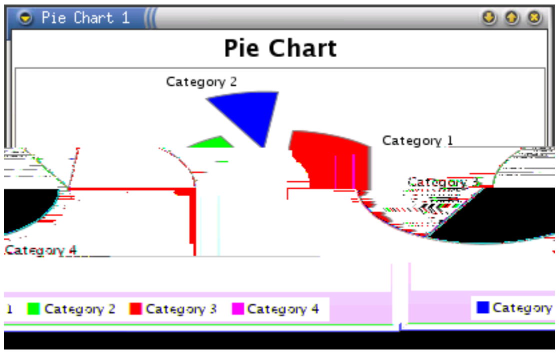
This section shows some sample charts created using the JFreeChart demonstration application. It is intended to give a reasonable overview of the types of charts that JFreeChart can generate.

2.2 Pie Charts

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



Individual pie sections can be "exploded", and the chart can take on an elliptical shape, as shown in the next example:

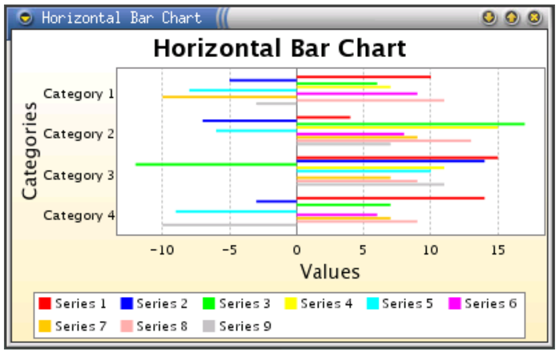


The original pie chart implementation was contributed by Andrzej Porebski.

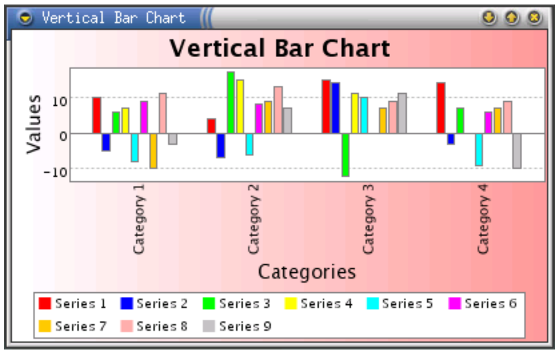
2.3 Bar Charts

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

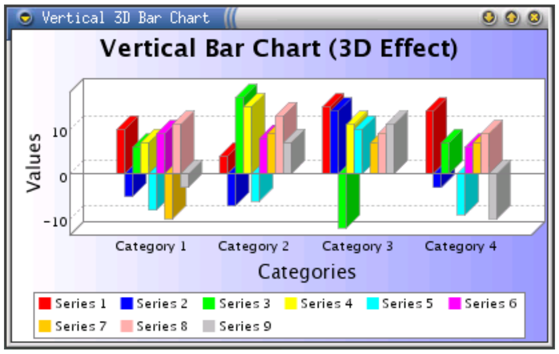
The first example is a *horizontal bar chart*:



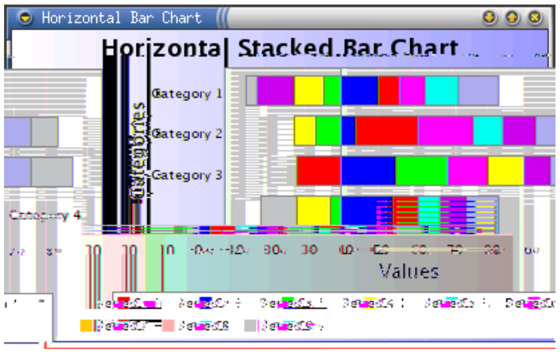
Using exactly the same data, but changing the orientation, we can generate a *vertical bar chart*:



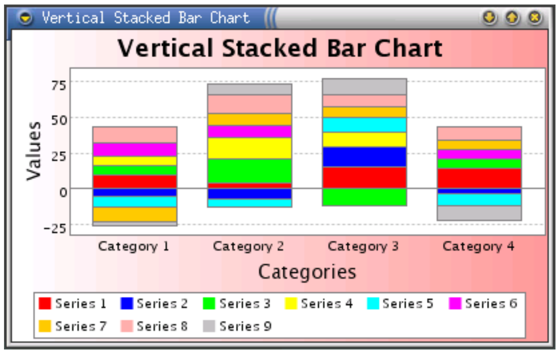
Vertical bar charts can be displayed with a 3D effect (thanks to Serge Grachov):



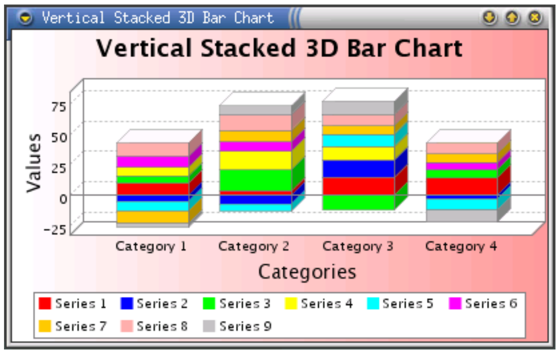
The bars can be stacked in a *stacked horizontal bar chart*:



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

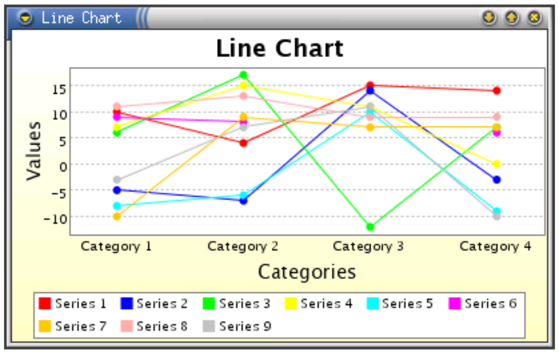


The stacked vertical bar chart can be displayed with a 3D effect (again thanks to Serge Grachov):



2.4 Line Chart

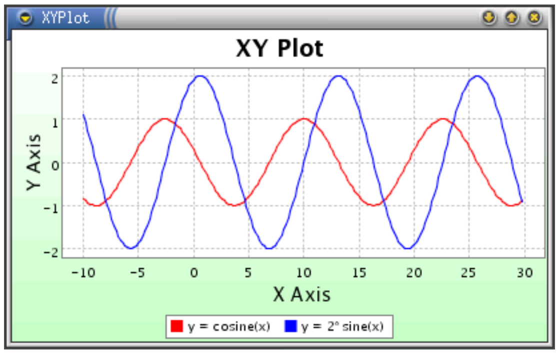
The *line chart* is generated using the same `CategoryDataset` that is used for the bar charts:


Line Chart
Line Chart
Values
15
10
5
0
-5
-10
Category 1
Category 2
Category 3
Category 4
Categories
🟥 Series 1 🟦 Series 2 🟩 Series 3 🟨 Series 4 Series 5 Series 6
Series 7 Series 8 Series 9

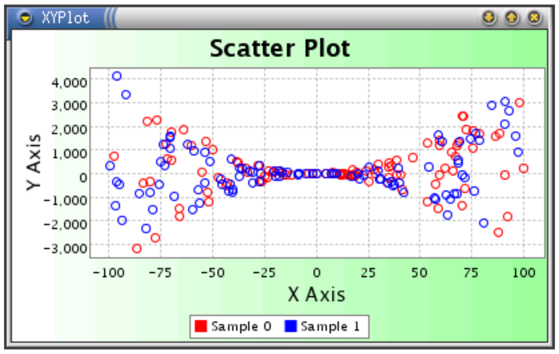
The data is the same, but the *line chart* gives you another presentation option.

2.5 XY Plots

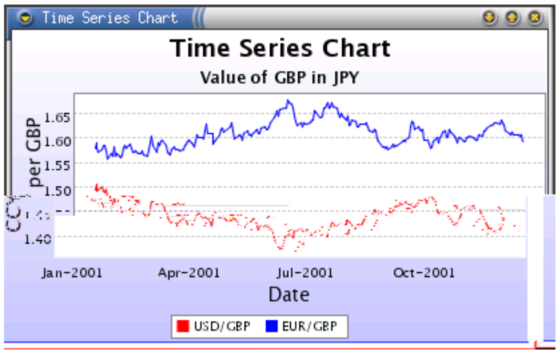
A third type of dataset, the `XYDataset`, is used to generate further chart types. The standard *XY plot* has numerical x and y axes. By default, lines are drawn between each data point:



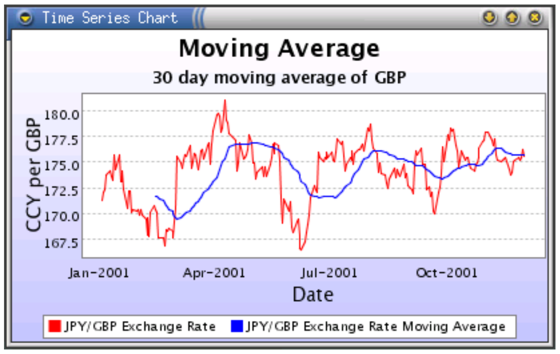
Shapes can be drawn at data points, rather than drawing lines between data points, for a *scatter plot*:



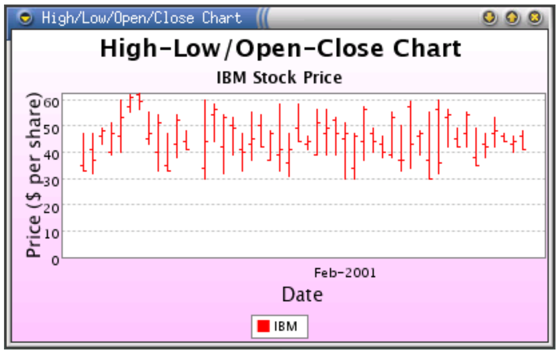
JFreeChart supports *time series charts*:



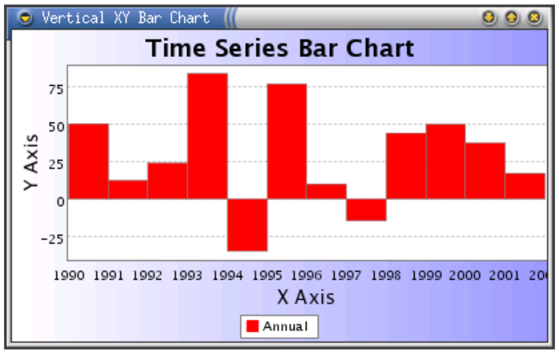
It is possible to add a moving average line to a time series plot:



You can display *high-low-open-close* data (thanks to Andrzej Porebski), using `HighLowDataset` (an extension of `XYDataset`):

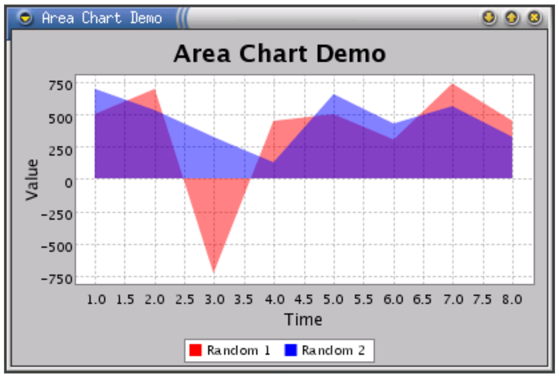


Bar charts over a numerical domain can be drawn using `IntervalXYDataset` (another extension of `XYDataset`):



2.6 Area Charts

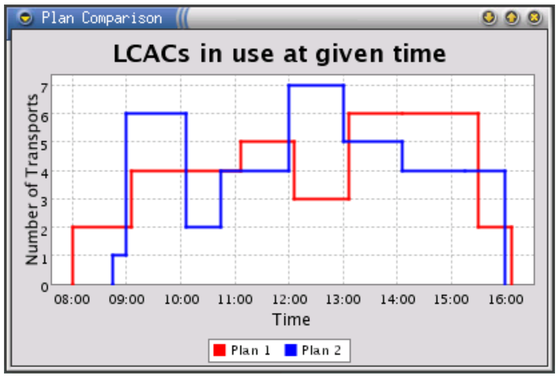
You can generate an *area chart* for data in a `CategoryDataset` or an `XYDataset`. The following example uses the latter:



The renderer classes for area charts were developed by Jon Iles and Hari.

2.7 Step Chart

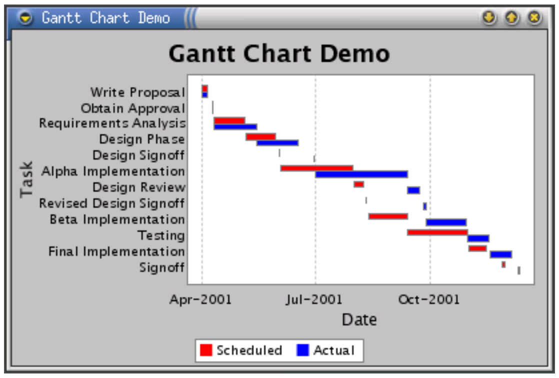
Here is an example of a *step chart*:



The renderer class for this chart was contributed by Roger Studner.

2.8 Gantt Chart

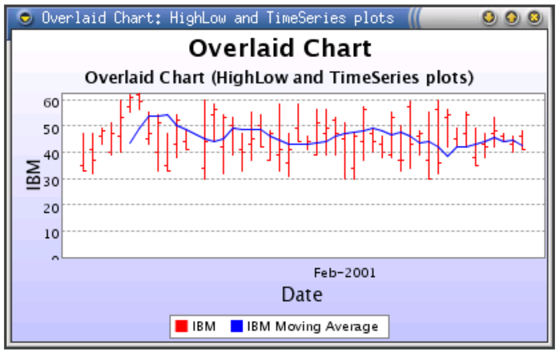
Simple *Gantt charts* can be generated using data from an `IntervalCategoryDataset`:



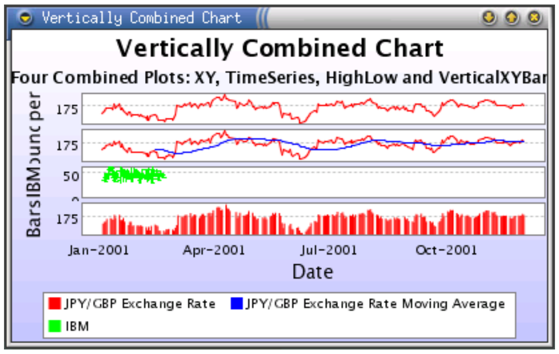
The renderer for this chart was developed by Eduard Martinescu.

2.9 Combined Charts

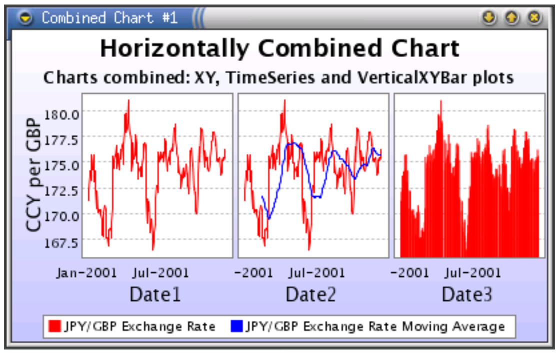
Bill Kelemen has extended JFreeChart to allow for combined charts, including overlaid charts:


Overlaid Chart: HighLow and TimeSeries plots
Overlaid Chart
Overlaid Chart (HighLow and TimeSeries plots)
IBM
60
50
40
30
20
10
0
Feb-2001
Date
🟥 IBM 🟦 IBM Moving Average

...horizontally combined charts:


Vertically Combined Chart
Vertically Combined Chart
Four Combined Plots: XY, TimeSeries, HighLow and VerticalXYBar
BarsIBMJunc per
175
175
50
175
Jan-2001
Apr-2001
Jul-2001
Oct-2001
Date
🟥 JPY/GBP Exchange Rate 🟦 JPY/GBP Exchange Rate Moving Average
🟩 IBM

...and vertically combined charts:


Combined Chart #1
Horizontally Combined Chart
Charts combined: XY, TimeSeries and VerticalXYBar plots
CCY per GBP
180.0
177.5
175.0
172.5
170.0
167.5
Jan-2001
Jul-2001
Date1
-2001
Jul-2001
Date2
-2001
Jul-2001
Date3
🟥 JPY/GBP Exchange Rate 🟦 JPY/GBP Exchange Rate Moving Average

2.10 Future Development

Given the open development model of JFreeChart, 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 web-page for updates.

3 Downloading and Installing JFreeChart

3.1 Introduction

This section contains instructions for downloading, unpacking, and recompiling JFreeChart (recompiling is optional, as the runtime jar files are included in the download). 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.object-refinery.com/jfreechart/index.html
```

There are two versions of the JFreeChart download:

File:	Description:
jfreechart-0.9.1.tar.gz	JFreeChart for Linux/Unix.
jfreechart-0.9.1.zip	JFreeChart for Windows.

The two files contain the same source code. All the text files in the Windows download have been recoded into DOS format (both carriage return *and* line-feed at the end of each line).

JFreeChart uses the JCommon Class Library (currently version `0.6.2`). 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.object-refinery.com/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 will be created in the same location as the download file.

3.3.1 Unpacking on Linux/Unix

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

```
tar xvzf jfreechart-0.9.1.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.1`.

3.3.2 Unpacking on Windows

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

```
jar -xvf jfreechart-0.9.1.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.9.1`.

3.3.3 The Files

The top-level directory (`jfreechart-0.9.1`) contains two files and three subdirectories, as described in the following table:

File/Directory:	Description:
`jars`	A directory containing the JFreeChart and JCommon run-time jar files.
`licence-LGPL.txt`	The licence for JFreeChart.
`README`	Important information - *read this first!*
`servlet`	A directory containing files required for the servlet demonstration.
`source`	A directory containing the source code for JFreeChart.

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

3.4 Running the Demonstration Application

A demonstration application is 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 application. All the classes are precompiled in the jar files.

To run the demo, type the following command[2] all on one line:

```
java -classpath jcommon-0.6.2.jar:jfreechart-0.9.1.jar:
jfreechart-0.9.1-demo.jar com.jrefinery.chart.demo.JFreeChartDemo
```

Depending on your system setup, you may need to specify the full path for the `java` executable. You may also need to type the full (or relative) path to the JFreeChart and JCommon jar files.

3.5 Compiling the Source

You can recompile the source files (contained in the `source` folder) using the `javac` tool, although I would recommend that you set up a project in your favourite development environment.

Nevertheless, if you insist upon using the command line...change to the `source` directory, then type the following command:

[2] If you are using Windows, you should use a semi-colon rather than a colon to separate the jar files.

```
javac -g:none -O -verbose -classpath .:../jars/jcommon-0.6.2.jar
com/jrefinery/chart/demo/JFreeChartDemo.java
```

This compiles the demonstration application *and most of the JFreeChart classes* (`javac` compiles each class for which it cannot find a `.class` file provided that it can find the corresponding `.java` source file). The class files are written to the same directories as the source files.

With the introduction of resource bundles for internationalisation, which are dynamically loaded by class name rather than directly referenced in code, you now need to separately compile the resource bundle classes. Type the following command:

```
javac -g:none -O -verbose -classpath .:../jars/jcommon-0.6.2.jar
com/jrefinery/chart/demo/resources/*.java
```

This compiles each of the resource bundle classes individually. You should now be able to run the `JFreeChartDemo` class.

There are a range of other demonstration applications alongside `JFreeChartDemo`. These can be compiled using a similar command:

```
javac -g:none -O -verbose -classpath .:../jars/jcommon-0.6.2.jar
com/jrefinery/chart/demo/XXX.java
```

Replace the text *XXX* with the name of the class you wish to compile.

Note that the `JFreeChartServletDemo` will not compile unless you have the `servlet.jar` file on the classpath—the file is included with Tomcat, and I'm guessing other servlet engines also.

3.6 Generating the Javadoc Documentation

The JFreeChart source code contains comprehensive *Javadoc comments*. You can use the `javadoc` tool to generate HTML documentation files directly from the source code.

To generate the documentation, use the `javadoc` utility as follows:

```
javadoc -sourcepath <your-source-directory> -d <your-output-directory>
com.jrefinery.chart com.jrefinery.chart.entity
com.jrefinery.chart.event com.jrefinery.chart.tooltips
com.jrefinery.chart.ui
```

There is a link to the Javadoc HTML pages on the JFreeChart web page.

4 Developing with JFreeChart

4.1 Overview

This section presents a tutorial on how to use the *JFreeChart* class library in your own projects.

4.2 The Basic Structure

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 a *graphics device*.[3]

In broad terms, JFreeChart achieves this by obtaining data from a Dataset, and delegating the drawing to a Plot object (which, in turn, delegates the drawing of individual data items to a CategoryItemRenderer or a XYItemRenderer, depending on the plot type).

The JFreeChart class can work with many different Dataset implementations, and even more Plot subclasses. The following table summarises the combinations that are currently available:

Dataset:	**Compatible Plot Types:**
PieDataset	PiePlot.
CategoryDataset	CategoryPlot subclasses with various renderers.
XYDataset	XYPlot with various renderers.
IntervalXYDataset	XYPlot with a VerticalXYBarRenderer.
HighLowDataset	XYPlot with a HighLowRenderer.
CandleStickDataset	XYPlot with a CandleStickRenderer.

There are a lot of combinations, but don't worry, just keep in mind that a chart usually has one Dataset and one Plot.

4.3 Creating Your First Chart

To illustrate, let's create a pie chart. First, we need to create a dataset that implements the PieDataset interface. The DefaultPieDataset class in the JCommon Class Library[4] is designed just for this purpose:

[3]Java supports several graphics devices—including the screen, the printer, and buffered images—via different implementations of 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 of the Batik Project).

[4]I moved all the dataset classes out of JFreeChart and into JCommon to underline the fact that the data classes are not intended just for generating charts—you ought to be able to use them in other ways. The TimeSeriesTableModel class is one example, making it easy for a time series to be displayed in a JTable

```
// 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));
```

Next, we need to create a chart. A convenient way to do this in JFreeChart is to use the `ChartFactory` class:

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

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

Now we have a chart, but we don't yet have anywhere to draw it. Let's create a frame to display the chart in. The `ChartFrame` class contains the machinery required to display charts:

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

And that's all there is to it...here is the complete program, so that you know which packages you need to import:

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

import com.jrefinery.data.DefaultPieDataset;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFrame;

public class First {

    public static void main(String[] args) {

        // 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);

        // create and display a frame...
        ChartFrame frame = new ChartFrame("Test", 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...

4.4 More about Datasets

In the previous section, we used the `DefaultPieDataset` class to supply data for our chart. JFreeChart can work with this class, because it implements the `PieDataset` interface. Take a look at this interface now, by looking at the source code[5] or the Javadoc HTML pages for the JCommon Class Library, or in the reference section towards the end of this document.

All of the datasets used by JFreeChart are defined by interfaces. This allows you to implement your own dataset using whatever data structures make sense for your own project. Of course, there are default classes available (in the JCommon Class Library) that implement each of the interfaces used by JFreeChart. You are free to use these default implementations if that is easier for you.

The `CategoryDataset` interface is used to access categorical data, most frequently used to display bar charts. In this dataset, the domain is a set of categories represented by any `java.lang.Object`. The categories are required to be unique (they are used to access the data values) and the `toString()` method is used to generate category labels. You'll probably find it convenient to use the `String` class for your categories.

The range for a `CategoryDataset` is numerical, with values represented by `Number` objects. You can use `null` values to represent missing or unknown data.[6]

The `XYDataset` interface is used to access data values in the form of (x, y) pairs. The domain values (x-values) are always numbers, even though sometimes they will be presented in a chart as dates. The range values (y-values) are always numbers too.

The `CategoryDataset` and `XYDataset` interfaces are not interchangeable. If a chart requires one type of data, you cannot substitute the other.

[5] One of the many advantages of free or open source software is that you can always refer to the source code to find out how things work.

[6] Most chart types check for null values. It is possible that some code is still missing this—if you get a null pointer exception due to null values in your dataset, please post a bug report.

5 Customising Charts

5.1 Introduction

This section describes common ways to customise the charts you create with JFreeChart. As far as possible, JFreeChart tries to use sensible default values when it creates charts. But at the same time, everything is defined to be configurable so that you can have complete control over the appearance of your charts.

5.2 Customising Charts

5.2.1 Adding Chart Titles

Charts are created with only one title (or sometimes no title at all). To add another title to your chart, use the `addTitle(...)` method. This method requires you to supply a reference to an `AbstractTitle` subclass, for example `TextTitle`:

```
TextTitle title = new TextTitle("New Chart Title");
myChart.addTitle(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.

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

5.2.2 Modifying Chart Titles

To modify a title that has already been added to a chart, you need to get a reference to the title. You can use the `getTitle(int)` method in the JFreeChart class:

```
AbstractTitle title = myChart.getTitle(titleIndex);
```

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

5.2.3 Setting the Background Color

You can use the `setBackgroundPaint(...)` method to set the background color for a chart. 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.

5.2.4 Using a Background Image

You can use the `setBackgroundImage(...)` method to set a background image for a chart. The image will be scaled to fit the area that the chart is being drawn into. You can also control the alpha-transparency for the image using the `setBackgroundImageAlpha(...)` method.

If you want an image to fill only the data area in your chart, then you need to add a background image to the `Plot` (described later).

5.2.5 Antialiasing

JFreeChart makes use of the Java2D antialiasing feature to draw smooth looking charts. You can switch this feature on or off as follows:

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

By default, charts are drawn with anti-aliasing.

5.3 Customising Plots

5.3.1 Overview

Much of the work in drawing a chart is delegated to the `Plot` class (or to a specific subclass of `Plot`). Often you will need to access this delegate in order to change the appearance of your chart. The `getPlot()` method in the `JFreeChart` class returns a reference to the plot being used by the chart.

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

You may need to cast this reference to a specific subclass of `Plot`. This is discussed later.

5.3.2 Setting the Background Paint

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

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

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

5.3.3 Using a Background Image

You can use the `setBackgroundImage(...)` method to set a background image for a plot. The image will be scaled to fit the area that the plot is being drawn into. You can also control the alpha-transparency for the image using the `setBackgroundAlpha(...)` method.

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

5.3.4 Changing Colors for Series

To change the colors used for the series in a plot, you should create an array of `Paint` objects:

```
Plot plot = myChart.getPlot();
Paint[] myPaintArray = new Paint[] { Color.red, Color.green, Color.blue
}; plot.setSeriesPaint(myPaintArray);
```

Ideally you should specify one `Paint` object per series, but JFreeChart will cycle through the array if there are too few items.

5.3.5 Other Properties

Some properties can only be changed after you have cast the result of the `getPlot()` method to an appropriate subclass of `Plot`. For example, if you want to set the gap before the first item in a `CategoryPlot`, you will need to use something like this:

```
CategoryPlot plot = (CategoryPlot)myChart.getPlot();
plot.setIntroGapPercent(0.10);
```

Refer to the documentation for the individual `Plot` subclasses for more information about the properties that you can change.

5.4 Customising Axes

5.4.1 Overview

Most plots in JFreeChart have two axes, the domain axis and the range axis, although some plots (for example, the `PiePlot` class) don't use axes at all. In the cases where axes are used, you can make many changes to the appearance of your chart by changing axis properties.

5.4.2 Obtaining an Axis Reference

Before you can change the properties of an axis, you need to obtain a reference to the axis.

The plot classes `CategoryPlot` and `XYPlot` both have the methods `getDomainAxis()` and `getRangeAxis()`. These methods return a reference to a `ValueAxis`, except in the case of a `CategoryPlot` the `getDomainAxis()` method returns a `CategoryAxis`.

Here is an example:

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

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

There are many different subclasses of the `Axis` class. 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 it almost always is), then you can do the following:

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

5.4.3 Setting the Axis Label

You can change the axis label by calling the `setLabel(...)` method in the `Axis` class. If you would prefer not to have a label for your axis, then use `setLabel(null)`.

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

5.4.4 Rotating Axis Labels

For vertical axes (`VerticalCategoryAxis` and `VerticalNumberAxis`), the axis label can be drawn with a vertical orientation to save space (this is the default). You can control this setting with the `setVerticalLabel(boolean)` method.

5.4.5 Rotating Category Labels

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

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

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

5.4.6 Hiding Tick Labels

To hide the tick labels for an axis:

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

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

5.4.7 Hiding Tick Marks

To hide the tick marks for an axis:

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

Category axes do not have tick marks.

5.4.8 Setting the Tick Size

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

Alternatively, for a `NumberAxis` you can specify your own set of tick units from which the axis will automatically select an appropriate tick size. See the next section.

5.4.9 Specifying the Auto Tick Units

In the `NumberAxis` class, there is a method `setStandardTickUnits(TickUnits collection)` that allows you to supply your own set of tick units for the auto-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 0.5 or 0.25. There is a method in the `TickUnits` class that returns a set of standard integer tick units (look at the source code to see how to create your own):

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

6 Charts Using Category Datasets

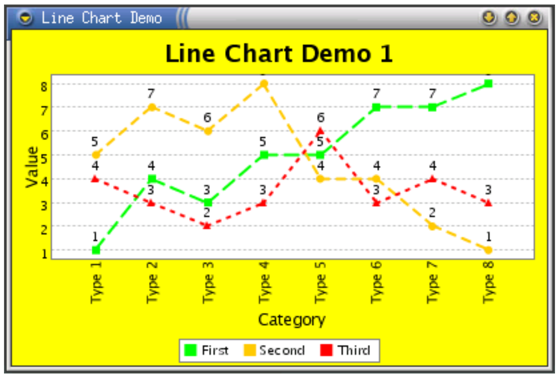
6.1 Introduction

This section describes how to generate charts based on data from the `CategoryDataset` interface.

6.2 Creating a Line Chart with Categorical Data

6.2.1 Overview

With JFreeChart, you can produce line charts using *categorical data* obtained via the `CategoryDataset` interface. In this section, I describe a sample application that creates the following line chart:



The full source code is included in the download.

6.2.2 The Dataset

You can use any implementation of the `CategoryDataset` interface to generate your chart. The `DefaultCategoryDataset` class is included with JFreeChart as a convenient implementation for those developers who do not wish to write their own datasets.

The code to create a dataset is relatively straightforward. Simply create a two-dimensional array of `double` values (each row in the array contains the data for one series, each column in the array contains the data for one category), then pass it to the appropriate constructor:

```
// create a dataset...
double[][] data = new double[][] {
    { 1.0, 4.0, 3.0, 5.0, 5.0, 7.0, 7.0, 8.0 },
    { 5.0, 7.0, 6.0, 8.0, 4.0, 4.0, 2.0, 1.0 },
    { 4.0, 3.0, 2.0, 3.0, 6.0, 3.0, 4.0, 3.0 }
};

DefaultCategoryDataset dataset = new DefaultCategoryDataset(data);
```

The `DefaultCategoryDataset` class will automatically generate names for the series and categories. You can specify these yourself, either in one of the alternative constructors, or using the `setSeriesNames(...)` and `setCategories(...)` methods.

```
// set the series names...
String[] seriesNames = new String[] { "First", "Second", "Third" };
dataset.setSeriesNames(seriesNames);

// set the category names...
String[] categories = new String[] { "Type 1", "Type 2", "Type 3", "Type 4",
                                     "Type 5", "Type 6", "Type 7", "Type 8"  };
dataset.setCategories(categories);
```

6.2.3 Constructing the Chart

The easiest way to construct the chart is to use the `ChartFactory` class:

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

This method constructs a plot with the appropriate axes and renderer, adds it to a chart, sets up the chart title and legend and returns a reference to the chart.

6.2.4 Customising the Line Chart

The default settings for the chart should produce an attractive chart, but of course you are free to modify any of the settings to change the appearance of the chart. In this example, we will make the following changes:

- change the chart background color;
- change the auto tick unit selection on the vertical axis so that the tick values always display integer values;
- change the series colors;
- change the series stroke (the pen/brush style used to draw the lines for each series);

Changing the chart's background color is simple:

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

To change the color used to represent each series, pass an array of `Paint` objects to the plot:

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

// set the color for each series...
plot.setSeriesPaint(new Paint[] { Color.green, Color.orange, Color.red });
```

The `plot` reference is retained for the remaining customisations.

You have full control over the line style for the plot. Simply create an array of `Stroke` objects and call the `setSeriesStroke(...)` method:

```
// set the stroke for each series...
Stroke[] seriesStrokeArray = new Stroke[3];
seriesStrokeArray[0] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 10.0f, 6.0f }, 0.0f);
seriesStrokeArray[1] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 6.0f, 6.0f }, 0.0f);
seriesStrokeArray[2] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 2.0f, 6.0f }, 0.0f);
plot.setSeriesStroke(seriesStrokeArray);
```

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(TickUnits.createIntegerTickUnits());
```

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

6.2.5 The Complete Program

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

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

import java.awt.Paint;
import java.awt.Color;
import java.awt.Stroke;
import java.awt.BasicStroke;
import com.jrefinery.data.CategoryDataset;
import com.jrefinery.data.DefaultCategoryDataset;
import com.jrefinery.ui.ApplicationFrame;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.ChartPanel;
import com.jrefinery.chart.CategoryPlot;
import com.jrefinery.chart.Axis;
import com.jrefinery.chart.HorizontalCategoryAxis;
import com.jrefinery.chart.NumberAxis;
import com.jrefinery.chart.TickUnits;

/**
 * A simple demonstration application showing how to create a line chart using data from a
 * CategoryDataset.
 */
public class LineChartDemo1 extends ApplicationFrame {

    /** The data. */
    protected CategoryDataset data;

    /**
     * Default constructor.
     */
    public LineChartDemo1(String title) {

        super(title);
        // create a dataset...
        double[][] data = new double[][] {
            { 1.0, 4.0, 3.0, 5.0, 5.0, 7.0, 7.0, 8.0 },
            { 5.0, 7.0, 6.0, 8.0, 4.0, 4.0, 2.0, 1.0 },
            { 4.0, 3.0, 2.0, 3.0, 6.0, 3.0, 4.0, 3.0 }
        };

        DefaultCategoryDataset dataset = new DefaultCategoryDataset(data);

        // set the series names...
        String[] seriesNames = new String[] { "First", "Second", "Third" };
        dataset.setSeriesNames(seriesNames);

        // set the category names...
        String[] categories = new String[] { "Type 1", "Type 2", "Type 3", "Type 4",
                                             "Type 5", "Type 6", "Type 7", "Type 8"  };
        dataset.setCategories(categories);

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

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

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

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

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

        // set the color for each series...
        plot.setSeriesPaint(new Paint[] { Color.green, Color.orange, Color.red });

        // set the stroke for each series...
        Stroke[] seriesStrokeArray = new Stroke[3];
        seriesStrokeArray[0] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                               1.0f, new float[] { 10.0f, 6.0f }, 0.0f);
        seriesStrokeArray[1] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                               1.0f, new float[] { 6.0f, 6.0f }, 0.0f);
        seriesStrokeArray[2] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                               1.0f, new float[] { 2.0f, 6.0f }, 0.0f);
        plot.setSeriesStroke(seriesStrokeArray);

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

        HorizontalCategoryAxis domainAxis = (HorizontalCategoryAxis)plot.getDomainAxis();
        domainAxis.setVerticalCategoryLabels(true);
        // OPTIONAL CUSTOMISATION COMPLETED.

        // add the chart to a panel...
        ChartPanel chartPanel = new ChartPanel(chart);
        this.setContentPane(chartPanel);

    }

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

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

    }

}
```

7 Charts Using XYDatasets

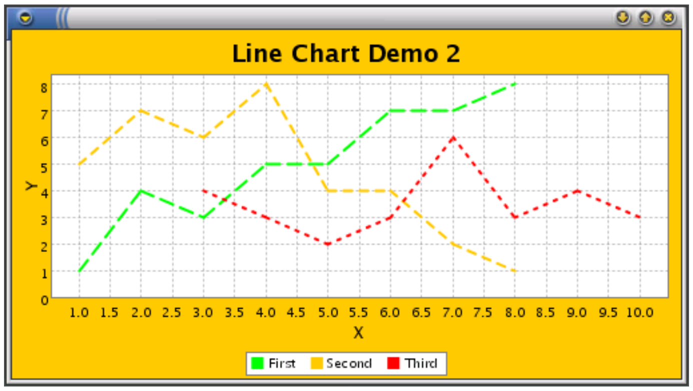
7.1 Introduction

This section describes how to create charts based on data from the `XYDataset` interface.

7.2 Creating a Line Chart with Numerical Data

7.2.1 Overview

With JFreeChart, you can produce line charts using *numerical data* obtained via the `XYDataset` interface. In this section, I describe a sample application that creates the following line chart:



The complete source code is available in the download.

7.2.2 The Dataset

You can use any implementation of the `XYDataset` interface to generate your chart. The `XYSeriesCollection` class is included with JFreeChart as a convenient implementation for those developers who do not wish to write their own datasets.

The code to create a dataset is relatively straightforward. Simply create each series individually, add them to a collection, and you have your dataset:

```
// 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);
```

7.2.3 Constructing the Chart

The easiest way to construct the chart is to use the `ChartFactory` class:

```
// create the chart...
JFreeChart chart = ChartFactory.createXYChart("Line Chart Demo 2",  // chart title
                                              "X",                  // domain axis label
                                              "Y",                  // range axis label
                                              dataset,              // data
                                              true                  // include legend
                                              );
```

This method constructs a plot with the appropriate axes and renderer, adds it to a chart, sets up the chart title and legend and returns a reference to the chart.

7.2.4 Customising the Line Chart

The default settings for the chart should produce an attractive chart, but of course you are free to modify any of the settings to change the appearance of the chart. In this example, we will make the following changes:

- change the chart background color;
- change the auto tick unit selection on the vertical axis so that the tick values always display integer values;
- change the series colors;
- change the series stroke (the pen/brush style used to draw the lines for each series);

Changing the chart's background color is simple:

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

To change the color used to represent each series, pass an array of `Paint` objects to the plot:

```
// get a reference to the plot for further customisation...
XYPlot plot = chart.getXYPlot();

// set the color for each series...
plot.setSeriesPaint(new Paint[] { Color.green, Color.orange, Color.red });
```

The `plot` reference is retained for the remaining customisations.

You have full control over the line style for the plot. Simply create an array of `Stroke` objects and call the `setSeriesStroke(...)` method:

```
// set the stroke for each series...
Stroke[] seriesStrokeArray = new Stroke[3];
seriesStrokeArray[0] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 10.0f, 6.0f }, 0.0f);
seriesStrokeArray[1] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 6.0f, 6.0f }, 0.0f);
seriesStrokeArray[2] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                       1.0f, new float[] { 2.0f, 6.0f }, 0.0f);
plot.setSeriesStroke(seriesStrokeArray);
```

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(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 an XY plot.

7.2.5 The Complete Program

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

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

import java.awt.Paint;
import java.awt.Color;
import java.awt.Stroke;
import java.awt.BasicStroke;
import com.jrefinery.data.XYDataset;
import com.jrefinery.data.XYSeriesCollection;
import com.jrefinery.data.XYSeries;
import com.jrefinery.ui.ApplicationFrame;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.ChartPanel;
import com.jrefinery.chart.XYPlot;
import com.jrefinery.chart.NumberAxis;
import com.jrefinery.chart.TickUnits;

/**
 * A simple demonstration application showing how to create a line chart using data from an
 * XYDataset.
 */
public class LineChartDemo2 extends ApplicationFrame {
```

```
/** The data. */
protected XYDataset data;

/**
 * Default constructor.
 */
public LineChartDemo2(String title) {

    super(title);

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

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

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

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

    // create the chart...
    JFreeChart chart = ChartFactory.createXYChart("Line Chart Demo 2",  // chart title
                                                  "X",                  // domain axis label
                                                  "Y",                  // range axis label
                                                  dataset,              // data
                                                  true                  // include legend
                                                  );

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

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

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

    // set the color for each series...
    plot.setSeriesPaint(new Paint[] { Color.green, Color.orange, Color.red });

    // set the stroke for each series...
    Stroke[] seriesStrokeArray = new Stroke[3];
    seriesStrokeArray[0] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                           1.0f, new float[] { 10.0f, 6.0f }, 0.0f);
    seriesStrokeArray[1] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                           1.0f, new float[] { 6.0f, 6.0f }, 0.0f);
    seriesStrokeArray[2] = new BasicStroke(2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                                           1.0f, new float[] { 2.0f, 6.0f }, 0.0f);
    plot.setSeriesStroke(seriesStrokeArray);

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

    // OPTIONAL CUSTOMISATION COMPLETED.

    // add the chart to a panel...
    ChartPanel chartPanel = new ChartPanel(chart);
    this.setContentPane(chartPanel);

}

/**
 * Starting point for the demonstration application.
 */
public static void main(String[] args) {
        LineChartDemo2 demo = new LineChartDemo2("Line Chart Demo 2");
        demo.pack();
        demo.setVisible(true);

    }

}
```

8 Combined Charts

8.1 Introduction

The combined charts facility was contributed to the JFreeChart project by Bill Kelemen. It provides a flexible mechanism for combining multiple plots on a single chart.

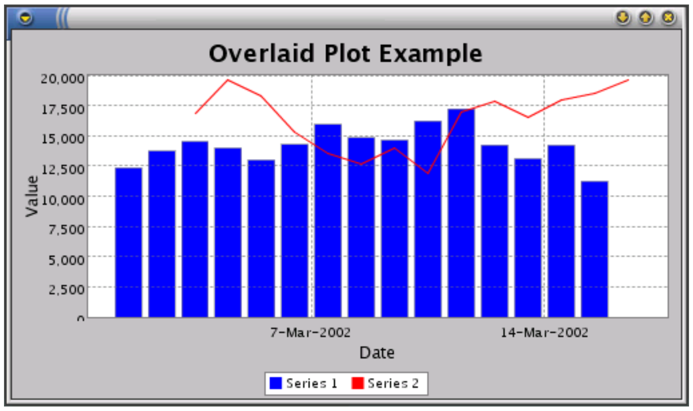
Since Bill first contributed the code, other changes have been made to JFreeChart. Most importantly, the dataset is now referenced by the `Plot` class rather than the `JFreeChart` class. This has made it possible to reorganise the combined charts code to make it easier to use.

In this section, I describe a few examples that use the combined charts facility. These examples are included in the JFreeChart download, so you can compile and run the code yourself.

8.2 Creating an Overlaid XY Plot

8.2.1 Overview

An *overlaid XY plot* is a specialised type of plot that combines two or more `XYPlot` instances together on one chart, using shared axes. Here I describe an example (included in the download) that displays a *vertical XY bar plot* combined with a *time series plot*:



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

8.2.2 The Application

The demonstration application is called `OverlaidXYPlotDemo`, and can be found in the `com.jrefinery.chart.demo` package.

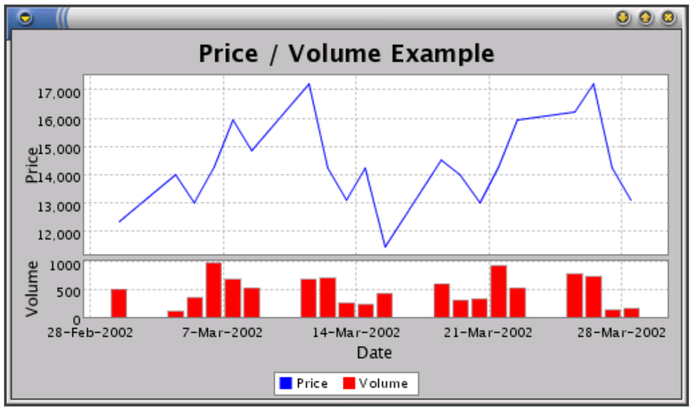
The `TimeSeriesCollection`

8.3 Creating a CombinedXYPlot

8.3.1 Overview

A *combined XY plot* is a plot that has two or more subplots sharing either the horizontal or the vertical axis.

To demonstrate, I have created a *price-volume chart*. This is a common type of chart used in the finance industry. It is used to plot the price of some commodity, along with the commodity's trading volume (the number of units traded, usually per day).



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

8.3.2 The Application

As in the previous overlaid plot example, I have used the `TimeSeriesCollection` class to represent both the price dataset and the volume dataset for this example. These datasets are assigned (in the example) to the object references `priceData` and `volumeData`.

8.3.3 Constructing the Chart

With the two datasets `priceData` and `volumeData`, we can proceed to construct the combined chart.

```
// create subplot 1...
XYDataset priceData = this.createPriceDataset();
XYItemRenderer renderer1 = new StandardXYItemRenderer();
renderer1.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy", "0.00"));
NumberAxis axis = new VerticalNumberAxis("Price");
axis.setAutoRangeIncludesZero(false);
XYPlot subplot1 = new XYPlot(priceData, null, axis, renderer1);

// create subplot 2...
IntervalXYDataset volumeData = this.createVolumeDataset();
```

```
XYItemRenderer renderer2 = new VerticalXYBarRenderer(0.20);
renderer2.setToolTipGenerator(new TimeSeriesToolTipGenerator("d-MMM-yyyy", "0.00"));
XYPlot subplot2 = new XYPlot(volumeData, null, new VerticalNumberAxis("Volume"), renderer2);
```

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

To create the parent plot:

```
// make a combined plot...
CombinedXYPlot plot = new CombinedXYPlot(new HorizontalDateAxis("Date"),
                                         CombinedXYPlot.VERTICAL);
plot.add(subplot1, 3);
plot.add(subplot2, 1);

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

The combined plot is created with a `VERTICAL` orientation, which means that the sub-plots are stacked from top to bottom.

You can control the amount of space allocated to each plot by specifying a *weight* for each plot as you add them to the parent plot. The weights are totalled, and each plot is allocated space based on its weight as a percentage of the total. In the example above, the first subplot is allocated 3/4 of the space, and the second subplot is allocated 1/4 of the space.

9 Exporting Charts to Acrobat PDF

9.1 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.

9.2 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.

9.3 iText

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

The home page for iText is:

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

At the time of writing, the latest version of iText is `0.92`.

9.4 Graphics2D

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

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

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

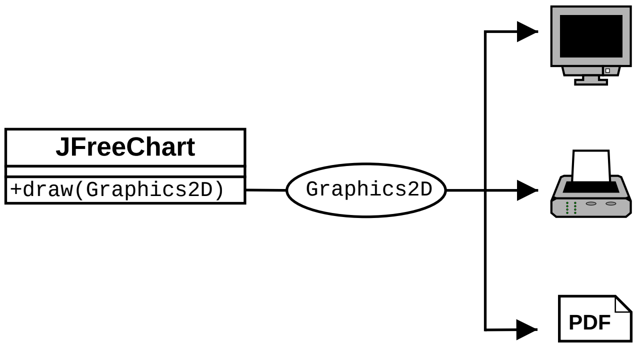


Figure 1: The JFreeChart `draw(...)` method

Recently, a new `PdfGraphics2D` class has been added to iText. This means that iText is now 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.

9.5 Getting Started

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

File:	Description:
jfreechart-0.9.1.jar	The JFreeChart class library.
jfreechart-0.9.1-demo.jar	The demo programs for JFreeChart (includes sample data).
jcommon-0.6.2.jar	The JCommon class library (used by JFreeChart).
iText-0.92.jar	The iText class library.

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

9.6 The Application

The first thing the sample application needs to do is create a chart. By making use of some of the sample data in the JFreeChart download, I will create a chart in just two lines of code:

```
// create a chart...
XYDataset data = DemoDatasetFactory.createSampleXYDataset();
JFreeChart chart = ChartFactory.createXYChart("PDF Chart 1", "X", "Y", data, true);
```

There is nothing special here—in fact you could replace these two lines of code 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("/home/dgilbert/jfreechart1.pdf");
saveChartAsPDF(fileName, chart, 400, 300);
```

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. You will need to replace the file name with something appropriate for your system. In a real application, you would provide some other means for the user to specify the filename, perhaps by presenting a file chooser dialog.

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

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

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

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

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

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

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

    document.close();

}
```

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

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

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

In the implementation of the `writeChartAsPDF(...)` method, I've chosen to create a PDF document with a custom page size (matching the requested size of the chart). You can easily adapt the code to use a different page size, alter the size and position of the chart and even draw multiple charts inside one PDF document.

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

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

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

}
```

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

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

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.File;
import java.io.OutputStream;
import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import com.lowagie.text.Document;
import com.lowagie.text.Rectangle;
import com.lowagie.text.DocumentException;
import com.lowagie.text.pdf.PdfWriter;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfTemplate;
import com.lowagie.text.pdf.FontMapper;
import com.lowagie.text.pdf.DefaultFontMapper;
import com.lowagie.text.pdf.BaseFont;
import com.jrefinery.data.XYDataset;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.demo.DemoDatasetFactory;

/**
```

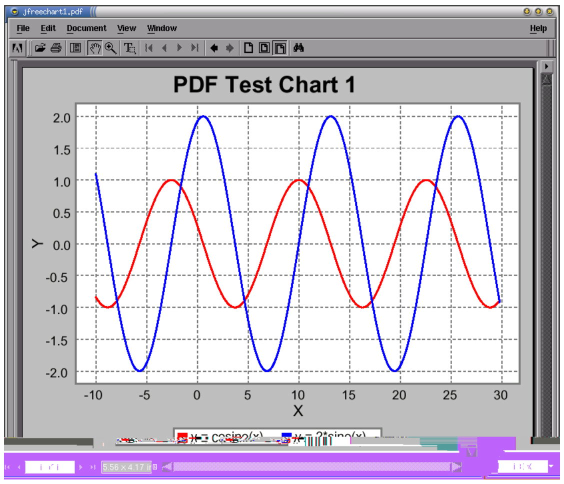
```
 * A simple demonstration showing how to write a chart to PDF format using
 * JFreeChart and iText.
 * <P>
 * You can download iText from http://www.lowagie.com/iText.
 */
public class ChartToPDFDemo1 {

    /**
     * Saves a chart to a PDF file.
     *
     * @param file The file.
     * @param chart The chart.
     * @param width The chart width.
     * @param height The chart height.
     */
    public static void saveChartAsPDF(File file,
                                      JFreeChart chart,
                                      int width, int height,
                                      FontMapper mapper) throws IOException {

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

    }

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

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

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

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

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

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

        document.close();

    }

    /**
     * Starting point for the demonstration application.
```

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

            try {
                // create a chart...
                XYDataset data = DemoDatasetFactory.createSampleXYDataset();
                JFreeChart chart = ChartFactory.createXYChart("PDF Test Chart 1",
                                                              "X", "Y",
                                                              data, true);

                // write the chart to a PDF file...
                File fileName = new File("/home/dgilbert/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 9.5 on your classpath.

9.7 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.

9.8 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.

9.8.1 Background

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

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

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

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

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

9.8.2 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.

9.8.3 Mapping Additional Fonts

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

```
/opt/jbuilder5/jdk1.3/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("/opt/jbuilder5/jdk1.3/jre/lib/fonts");
DefaultFontMapper.BaseFontParameters pp =
                              mapper.getBaseFontParameters("Arial Unicode MS");
if (pp!=null) {
    pp.encoding = BaseFont.IDENTITY_H;
}
```

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

```
// create a chart...
XYDataset data = DemoDatasetFactory.createTimeSeriesCollection2();
JFreeChart chart = ChartFactory.createTimeSeriesChart("PDF Test",
                                                      "Time", "Price",
                                                      data, true);
String text = "\u278A\u20A0\u20A1\u20A2\u20A3\u20A4\u20A5\u20A6\u20A7\u20A8\u20A9";
Font font = new Font("Arial Unicode MS", Font.PLAIN, 12);
TextTitle subtitle = new TextTitle(text, font);
chart.addTitle(subtitle);
```

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

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

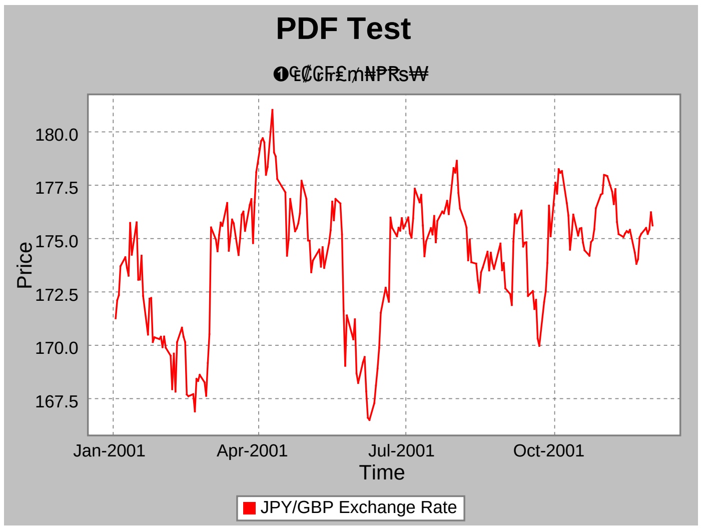


Figure 2: A Unicode subtitle

10 Exporting Charts to SVG Format

10.1 Introduction

In this section, I describe how to export a chart to a file in SVG format, using JFreeChart and Batik.

10.2 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).

10.3 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 version of Batik is `1.1.1`.

10.4 Batik and JFreeChart

Getting JFreeChart to work with Batik is relatively painless. I've only spent a limited amount of time working with Batik, so I'm no expert, but here I will describe a simple program that creates a chart and saves it in SVG format in a file. Hopefully this will be enough to get you started.

10.5 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-0.6.2.jar`	Common classes from The Object Refinery.
`jfreechart-0.9.1.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.

10.6 The Application

Create a project in your favourite Java development environment, and type in the following program:

```
package svgtest;

import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.data.DefaultPieDataset;

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

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

public class Application {

    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);

        // THE FOLLOWING CODE BASED ON THE EXAMPLE IN THE BATIK DOCUMENTATION...
        // Get a DOMImplementation
        DOMImplementation domImpl = GenericDOMImplementation.getDOMImplementation();

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

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

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

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

    }

}
```

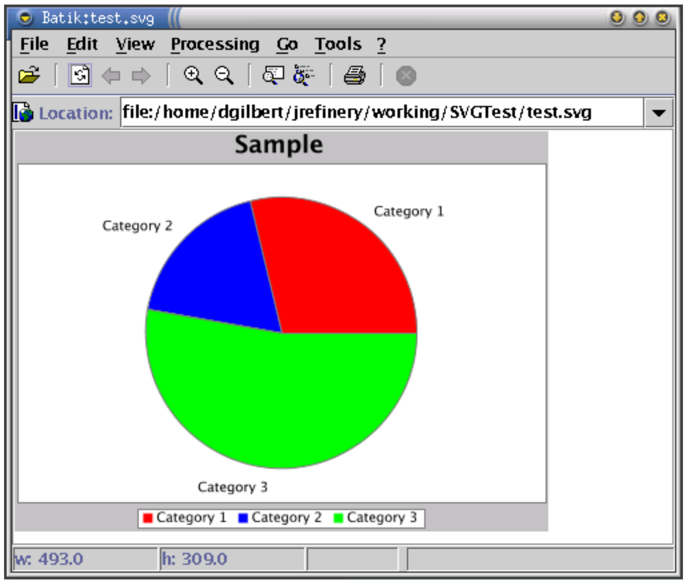
Running this program creates a file `test.svg` in SVG format.

10.7 Viewing the SVG

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

```
    jar -jar batik-svgbrowser.jar
```

The following screen shot shows the pie chart that we created earlier, displayed in the Batik browser application:


Batik:test.svg
File Edit View Processing Go Tools ?
Location: file:/home/dgilbert/jrefinery/working/SVGTest/test.svg
Sample
Category 1
Category 2
Category 3
🟥 Category 1 🟦 Category 2 🟩 Category 3
w: 493.0
h: 309.0

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

11 Packages

11.1 Overview

The following sections contain reference information for the packages that make up JFreeChart.

Package (`com.jrefinery.*`):	**Description:**
`chart`	The main chart classes.
`chart.data`	Some data fitting classes (to be moved).
`chart.entity`	Classes representing chart entities.
`chart.event`	The event classes.
`chart.junit`	Tests for the JFreeChart library based on the JUnit framework
`chart.tooltips`	The tooltip classes.
`chart.ui`	User interface classes.
`chart.demo`	The demonstration application.
`chart.demo.jdbc.servlet`	A servlet demonstration.
`chart.demo.jdbc.swing`	A JDBC demonstration.
`chart.demo.resources`	Resource bundles for user interface items that require localisation.

I also include documentation for the `com.jrefinery.data` package—part of the JCommon class library—since it is used extensively by JFreeChart.

Additional information can be found in the Javadoc HTML files for JFreeChart and JCommon.

12 Package: com.jrefinery.chart

12.1 Overview

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

12.2 AbstractCategoryItemRenderer

12.2.1 Overview

A base class that can be used to implement a new *category item renderer*.

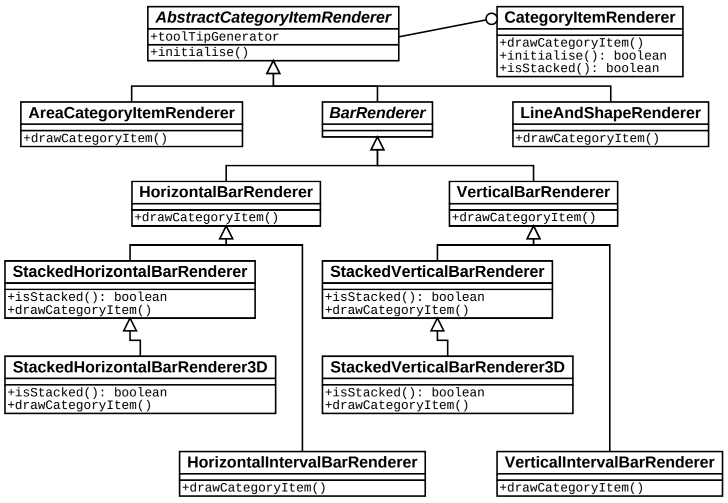


Figure 3: Category item renderers

12.2.2 Constructors

The default constructor:

> `protected AbstractCategoryItemRenderer();`
> Creates a new renderer with a standard tool tip generator. The tool tip generator is set up even if it is never used.

The other constructor allows you to supply a custom tool tip generator:

> `protected AbstractCategoryItemRenderer(CategoryToolTipGenerator toolTipGenerator);`
> Creates a new renderer with a custom tool tip generator.

12.2.3 Methods

The following method is called once every time the chart is drawn:

> `public void initialise(...);`
> Performs any initialisation required by the renderer. The default implementation simply stores a local reference to the `info` object (which may be `null`).

12.2.4 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

See Also
`CategoryItemRenderer`.

12.3 AbstractTitle

12.3.1 Overview

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

The `JFreeChart` class maintains a list of titles, which can hold zero, one or many titles.

12.3.2 Constructors

The standard constructor:

> `protected AbstractTitle(int position, int horizontalAlignment, int verticalAlignment, Spacer spacer);`
> Creates a new `AbstractTitle`.

12.3.3 Notes

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

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

See Also
`ImageTitle`, `TextTitle`.

12.4 AbstractXYItemRenderer

12.4.1 Overview

A convenient base class for creating new `XYItemRenderer` implementations.

This class provides a property change mechanism to support the requirements of the `XYItemRenderer` interface.

12.4.2 Constructors

This class provides a default constructor which allocates storage for the list of property change listener references.

12.4.3 Methods

To register a `PropertyChangeListener` with the renderer:

> `public void addPropertyChangeListener(PropertyChangeListener listener);`
> Registers a listener so that it receives notification of any changes to the renderer.

If an object no longer wishes to receive property change notifications:

> `public void removePropertyChangeListener(PropertyChangeListener listener);`
> Removes a listener so that it no longer receives notification of changes to the renderer.

See Also
`XYItemRenderer`, `XYPlot`.

12.5 AreaCategoryItemRenderer

12.5.1 Overview

A *category item renderer* that draws an area chart using data from a `CategoryDataset`. You can use this renderer with the `VerticalCategoryPlot` class.

12.5.2 Methods

This renderer overrides two methods from the superclass `AbstractCategoryItemRenderer`:

> `public void drawRangeMarker(...);`
> Draws a vertical line to represent a marker on the range axis.

12.5.3 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

See Also

CategoryItemRenderer.

12.6 AreaXYItemRenderer

12.6.1 Overview

A renderer that can be used by `XYPlot` to draw an *area chart*. An area chart is similar to a line chart, except that the region between the line and the x-axis is filled with a solid color.

12.6.2 Constructors

The default constructor sets up the renderer to draw area charts:

> `public AreaXYItemRenderer();`
> Creates a new `AreaXYItemRenderer`. By default, the type is set to `AREA` (see the next constructor).

You can change the appearance of the chart by specifying the type:

> `public AreaXYItemRenderer(int type);`
> Creates a new `AreaXYItemRenderer` using one of the following types: `SHAPES`, `LINES`, `SHAPES_AND_LINES`, `AREA`, `AREA_AND_SHAPES`.

12.6.3 Notes

You can see from this second constructor that the `AreaXYItemRenderer` class is based on 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 `AreaXYItemRenderer` with `StandardXYItemRenderer`.

See Also

StandardXYItemRenderer, XYItemRenderer, XYPlot.

12.7 Axis

12.7.1 Overview

An abstract class representing an axis (horizontal or vertical). Some subclasses of `Plot` will use axes to display data.

Figure 4 illustrates the axis class hierarchy.

12.7.2 Constructors

To create a new `Axis`:

> `protected Axis(String label);`
> Creates a new `Axis`, with the specified label.

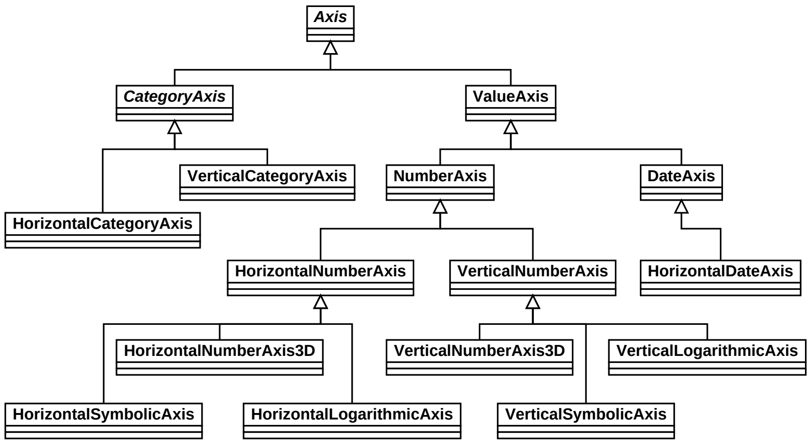


Figure 4: Axis classes

12.7.3 Attributes

The `Axis` class has the following attributes:

Attribute:	**Description:**
`Plot`	The plot that the axis belongs to.
`Label`	The axis label.
`LabelFont`	The font for the axis label.
`LabelPaint`	The color for the axis label.
`LabelInsets`	The space to leave blank around the axis label.
`TickLabelsVisible`	A flag controlling the visibility of tick labels.
`TickLabelFont`	The font for the tick labels.
`TickLabelPaint`	The color for the tick labels.
`TickLabelInsets`	The space to leave around the tick labels.
`TickMarksVisible`	A flag controlling the visibility of tick marks.
`TickMarkStroke`	The *stroke* used to draw the tick marks.

The following default values are used for attributes wherever necessary:

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);`

12.7.4 Notes

The `Axis` class implements a notification mechanism that informs registered listeners whenever a change is made to an axis. The following methods are used:

`public void addChangeListener(AxisChangeListener listener);`
Registers an object to receive notification whenever the axis changes.

`public void removeChangeListener(AxisChangeListener listener);`
Deregisters an object, so that it no longer receives notification when the axis changes.

`public void notifyListeners(AxisChangeEvent event);`
Notifies all registered listeners that a change has been made to the axis.

See Also
`AxisConstants`, `AxisChangeEvent`, `AxisChangeListener`, `AxisNotCompatibleException`.

12.8 AxisConstants

12.8.1 Overview

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

12.8.2 Notes

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

See Also
`Axis`.

12.9 AxisNotCompatibleException

12.9.1 Overview

An exception that indicates that an attempt has been made to assign an axis to a `Plot` where the axis is not compatible with the plot type (for example, a `VerticalCategoryAxis` will not work with an `XYPlot`).

12.9.2 Constructors

To create a new exception:

`public AxisNotCompatibleException(String message);`
Creates a new exception.

12.9.3 Notes

The `AxisNotCompatibleException` is a subclass of `RuntimeException`.

See Also
`PlotNotCompatibleException`.

12.10 BarRenderer

12.10.1 Overview

A base class that is used to implement various *category item renderers* that represent data using bars.

See Also

`HorizontalBarRenderer`, `VerticalBarRenderer`.

12.11 CandlestickRenderer

12.11.1 Overview

A renderer that is used by the `XYPlot` class to generate *candlestick charts*. This class implements the `XYItemRenderer` interface.

A recent addition to this renderer is the ability to represent volume information in the background of the chart.

12.11.2 Constructors

To create a new renderer:

```
public CandlestickRenderer(double candleWidth);
```
Creates a new renderer.

12.11.3 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.

12.11.4 Notes

This renderer requires a `HighLowDataset`.

The original candlestick chart was developed by Sylvain Vieujot. As JFreeChart evolved, I converted the code to a class that implements the `XYItemRenderer` interface. Sylvain has continued to enhance the renderer, recently incorporating a feature to display volume data in the background of the chart.

See Also
`XYItemRenderer`, `HighLowDataset`.

12.12 CategoryAxis

12.12.1 Overview

An abstract base class for axes that display labels for categorical data.

12.12.2 Notes

The `CategoryAxis` class extends the `Axis` class. Note that this class doesn't add anything to `Axis`—it occupies its place in the class hierarchy purely for descriptive purposes.

Known subclasses include `HorizontalCategoryAxis` and `VerticalCategoryAxis`.

See Also
`Axis`.

12.13 CategoryItemRenderer

12.13.1 Overview

The interface that must be supported by a *category item renderer*. A renderer is a plug-in for the `CategoryPlot` class that is responsible for drawing individual data items.

A number of different renderers have been developed, allowing different chart types to be generated easily.

The following table lists the renderers that have been implemented to date:

Class:	Description:
`HorizontalBarRenderer`	Represents data using horizontal bars (anchored at zero).
`VerticalBarRenderer`	Represents data using vertical bars (anchored at zero).
`HorizontalIntervalBar-Renderer`	Draws intervals using horizontal bars. This renderer can be used to create simple GANTT charts.
`LineAndShapeRenderer`	Draws lines and/or shapes to represent data.
`StackedHorizontalBar-Renderer`	Used to create a horizontal stacked bar chart.
`StackedVerticalBar-Renderer`	Used to create a vertical stacked bar chart.

Classes that implement the `CategoryItemRenderer` interface are expected to be immutable.[7] That way, you can only change the appearance of the chart by calling the `setRenderer(...)` method in the `CategoryPlot` class, and so the proper event notification can be triggered to update the chart.

12.13.2 Methods

The interface defines an initialisation method:

```
public void initialise(...);
```

This method is called at the start of every chart redraw. It gives the renderer a chance to precalculate any information it might require later when rendering individual data items.

For data range calculations, the `CategoryPlot` class needs to know whether or not the renderer stacks values. This can be determined via the following method:

```
public boolean isStacked();
```

Returns `true` if the values are stacked, and `false` otherwise.

The most important method is the one that actually draws a data item:

```
public Shape drawCategoryItem(...);
```

Draws one item on a category plot.

12.13.3 Notes

Classes that implement the `CategoryItemRenderer` interface are used by the `CategoryPlot` class. They cannot be used by the `XYPlot` class (which uses implementations of the `XYItemRenderer` interface).

See Also

`CategoryPlot`.

[7] This will change in a future version. A property change notification mechanism will be added, to parallel the same feature that has been added to the `XYItemRenderer` interface.

12.14 CategoryPlot

12.14.1 Overview

A base class that controls the drawing of a plot based on data from a `CategoryDataset`. The visual appearance of the plot can be customised by setting a `CategoryItemRenderer` for the plot.

12.14.2 Constructors

There are two constructors for `CategoryPlot`. The simpler of the two requires the caller to specify the axes and the renderer, with all other axis properties assuming default values:

```
protected CategoryPlot(Axis horizontalAxis, Axis verticalAxis,
CategoryItemRenderer renderer);
```

Creates a new `CategoryPlot` using mostly default values.

The alternative constructor allows the caller to specify a wide range of axis properties. Refer to the source code or Javadoc HTML pages for details.

12.14.3 Attributes

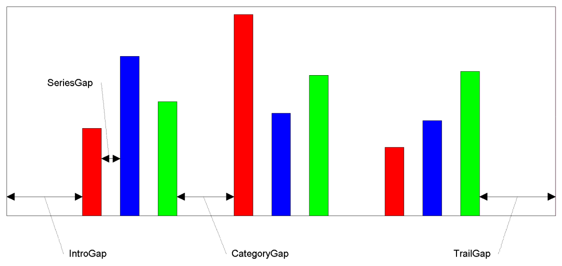
The `CategoryPlot` adds the following attributes to those that it inherits from the `Plot` class:

Attribute:	**Description:**
`Renderer`	The class responsible for rendering each data item in the plot.
`IntroGapPercent`	The space *before* the first item in the plot.
`TrailGapPercent`	The space *after* the last item in the plot.
`CategoryGapsPercent`	The space between the last item in one category, and the first item in the next category.
`ItemGapsPercent`	The space between two bars in the same category.
`ToolTipGenerator`	The tooltip generator (optional).

The following default values are used for attributes wherever necessary:

Name:	**Value:**
`DEFAULT_INTRO_GAP_PERCENT`	`0.05` (5 percent)
`DEFAULT_TRAIL_GAP_PERCENT`	`0.05` (5 percent)
`DEFAULT_CATEGORY_GAPS_PERCENT`	`0.20` (20 percent)
`DEFAULT_ITEM_GAPS_PERCENT`	`0.15` (15 percent)
`DEFAULT_TOOL_TIP_GENERATOR`	`null`

This diagram illustrates the purpose of the "gap" attributes:



12.14.4 Methods

You can control the appearance of the plot by setting a renderer for the plot. The renderer is responsible for drawing a visual representation of each data item:

> `public void setRenderer(CategoryItemRenderer renderer);`
> Sets the renderer for the plot. A range of different renderers are available. If you set the renderer to `null`, an empty chart is drawn.

To get a reference to the category axis for the plot:

> `public abstract CategoryAxis getDomainAxis();`
> Returns the category axis for the plot.

To get a reference to the numerical axis for the plot:

> `public abstract ValueAxis getRangeAxis();`
> Returns the value axis for the plot.

To set a tooltip generator for the plot:

> `public void setToolTipGenerator(CategoryToolTipGenerator generator);`
> Sets a tooltip generator for the plot. If tooltip information is requested at the time a chart is drawn, this generator will be used to create the text for each data item. Registering your own generator gives you full control over the tooltip text formatting.

A zoom method is provided to support the zooming function provided by the `JFreeChartPanel` 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.

12.14.5 Notes

The `CategoryDataset` interface is part of the JCommon Class Library.

A number of different item renderers have been implemented—see the listings in the entries for the subclasses `HorizontalCategoryPlot` and `VerticalCategoryPlot`.

See Also

`HorizontalCategoryPlot`, `VerticalCategoryPlot`.

12.15 CategoryPlotConstants

12.15.1 Overview

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

12.16 ChartFactory

12.16.1 Overview

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

12.16.2 Methods

```
public static JFreeChart createPieChart(String title, PieDataset data,
boolean legend);
```

Creates a *pie chart* for the given `PieDataset`.

```
public static JFreeChart createVerticalBarChart(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```

Creates a *vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createVerticalBarChart3D(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```

Creates a *vertical bar chart with 3D effect* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```

Creates a *stacked vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart3D(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```

Creates a *stacked vertical bar chart with 3D effect* for the given `CategoryDataset`.

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

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

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

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

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

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

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

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

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

12.16.3 Notes

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

See Also

`JFreeChart`.

12.17 ChartFrame

12.17.1 Overview

A frame containing chart within a `ChartPanel`.

12.17.2 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.

12.17.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`ChartPanel`.

12.18 ChartMouseEvent

12.18.1 Overview

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

12.18.2 Notes

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

See Also
`ChartPanel`, `ChartMouseListener`.

12.19 ChartMouseListener

12.19.1 Overview

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

12.19.2 Methods

There are two methods defined by this interface.

The first receives notification of mouse click events:

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

The second 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.

12.19.3 Notes

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

See Also

`ChartPanel`, `ChartMouseEvent`.

12.20 ChartPanel

12.20.1 Overview

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

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 the chart to a PNG format file;
- zooming options;

In addition, the panel can:

- provide offscreen buffering to improve performance when redrawing overlapping frames;
- display tooltips for some chart types;

All of these features are used in the demonstration application that is included with the JFreeChart distribution.

12.20.2 Constructors

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

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

By default, the panel is automatically updated whenever the chart changes.

12.20.3 Methods

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

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

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

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

The panel includes support for tooltips (which are available on most chart types). To turn this feature on or off, use the following method:

> `public void setToolTipGeneration(boolean flag);`
> Switches the tooltips feature on or off for this panel.

As the space available for drawing a chart gets smaller and smaller, it becomes more and more difficult to layout the components of the chart without overlaps. One solution to this is to specify the minimum drawing area for the chart—if the space on the panel is less than the minimum, then the chart is drawn in a buffer at the minimum size, then scaled into the available space on the panel. Use the following method to specify the minimum size:

> `public void setMinimumDrawArea(Rectangle2D area);`
> Sets the minimum size for drawing the chart. A scaling transformation is used to fit the chart into spaces smaller than this if required.

12.20.4 Notes

The panel includes support for displaying tooltips for a chart.

See Also

`JFreeChart`.

12.21 ChartPanelConstants

12.21.1 Overview

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

12.22 ChartRenderingInfo

12.22.1 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);
- entities (including tooltip information);

12.22.2 Constructors

The default constructor:

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

See Also

`EntityCollection`.

12.23 ChartUtilities

12.23.1 Overview

This class contains some useful methods for use with charts.

12.23.2 Methods

The methods include:

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

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

12.23.3 Notes

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

See Also

JFreeChart.

12.24 CombinedXYPlot

12.24.1 Overview

A subclass of `XYPlot` that allows you to combined multiple plots on one chart. The subplots share either the horizontal or vertical axis from the parent, and maintain one "non-shared" axis each.

Figure 5 illustrates the relationship between the parent plot and its subplots (in this case the combination is *vertical*).

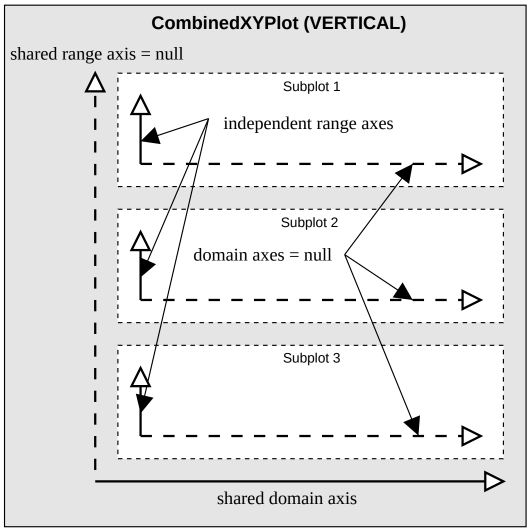


Figure 5: CombinedXYPlot axes

12.24.2 Methods

To add a subplot:

> `public void add(XYPlot subplot, int weight);`
> Adds a subplot. The subplot 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.

12.24.3 Notes

The dataset for this class should be set to `null` (only the subplots display data).

The subplots managed by this class should have one axis set to `null` (the shared axis is maintained by this class).

A demonstration of this type of plot is described in section 8.3.

See Also

`XYPlot`, `OverlaidXYPlot`.

12.25 CrosshairInfo

12.25.1 Overview

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

12.25.2 Constructors

The default constructor:

```
public CrosshairInfo();
```

Creates a `CrosshairInfo` object.

12.25.3 Methods

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

```
public void updateCrosshairPoint(double candidateX, double candidateY);
```

Creates a `CrosshairInfo` object.

12.26 DateAxis

12.26.1 Overview

The base class for axes that display date/time values—extends `ValueAxis`. This class is designed to be flexible about the range of dates/times that it can display—anything from several milliseconds to several decades should be handled.

12.26.2 Constructors

This class has two constructors—the first requires all properties to be specified, while the second assumes default values for many properties.

12.26.3 Attributes

`DateAxis` defines the following properties:

Attribute:	Description:
`minimumDate`	The minimum date (or time) visible on the axis.
`maximumDate`	The maximum date (or time) visible on the axis.
`tickUnits`	The `DateUnit` used for tick marks.
`tickLabelFormatter`	The `DateFormat` object used to format the tick labels.

12.26.4 Notes

In the current implementation, there is one subclass: `HorizontalDateAxis`, which can be used with an `XYPlot` to present time series data.

See Also

`HorizontalDateAxis`, `DateUnit`.

12.27 DateTitle

12.27.1 Overview

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

12.27.2 Notes

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

See Also

`AbstractTitle`.

12.28 DateUnit

12.28.1 Overview

Represents a fixed unit of time, used to specify the tick units for a `DateAxis`.

12.28.2 Constructors

There is just one constructor:

```
public DateUnit(int field, int count);
```

Creates a new `DateUnit`.

The `field` attribute uses constants defined in the `java.util.Calendar` class:

Time Unit:	Constant:
Year	`Calendar.YEAR`
Month	`Calendar.MONTH`
Day	`Calendar.DATE`
Hour	`Calendar.HOUR_OF_DAY`
Minute	`Calendar.MINUTE`
Second	`Calendar.SECOND`
Millisecond	`Calendar.MILLISECOND`

You should not use any of the other constants defined in `java.util.Calendar`.

12.28.3 Methods

The following method is used for simple date addition:

```
public Date addToDate(Date base);
```
Creates a new `Date` that is one `DateUnit` after the `base` date.

12.28.4 Notes

To create a `DateUnit` representing one week, use the following code:

```
DateUnit week = new DateUnit(Calendar.DATE, 7);
```

If you want to create a `DateUnit` measured in hours, note that a common mistake is to use the `Calendar.HOUR` constant in the constructor. This doesn't work—you should use `Calendar.HOUR_OF_DAY` instead.

See Also
`DateAxis`.

12.29 DefaultShapeFactory

12.29.1 Overview

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

12.30 HighLow

12.30.1 Overview

Represents one item used by a *HighLowRenderer* during the rendering process.

12.30.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

HighLowRenderer.

12.31 HighLowRenderer

12.31.1 Overview

A renderer that can be used with the `XYPlot` class and a `HighLowDataset` to create high-low-open-close charts.

12.31.2 Methods

Implements the `drawItem(...)` method defined in the `XYItemRenderer` interface.

12.31.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also

XYPlot, XYItemRenderer.

12.32 HorizontalAxis

12.32.1 Overview

An interface that *must* be implemented by all horizontal axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

12.32.2 Methods

The interface defines two methods. The plot will call one of these two methods, depending on the implementation.

```
public Rectangle2D reserveAxisArea(Graphics2D g2,
Plot plot, Rectangle2D drawArea, double reservedWidth);
```

Calculates the area that the horizontal axis requires to draw itself. If this method is used, it will be called after the vertical axis has determined the width that it requires—the argument `reservedWidth` contains this value.

```
public double reserveHeight(Graphics2D g2, Plot plot,
Rectangle2D drawArea);
```

Estimates the height that the horizontal axis requires to draw itself. If this method is used, it will be called before the vertical axis is asked to calculate the area that it requires—the height returned by this method will be passed to the vertical axis.

See Also
`VerticalAxis`.

12.33 HorizontalBarRenderer

12.33.1 Overview

A renderer that draws horizontal bars.

12.33.2 Methods

The plot calls the following method to draw each bar:

> `public Shape drawBar(...);`
> This method returns the y-coordinate of the center of the specified category. The category axis will call this method to determine where to place the category labels, because it has no knowledge of the distribution of categories (these could vary, depending on the nature of the plot).

12.33.3 Notes

The important methods from this class need to be factored out into an interface.

Refer to Javadoc HTML files and source code for details.

See Also
`StackedHorizontalBarRenderer`.

12.34 HorizontalCategoryAxis

12.34.1 Overview

A horizontal axis that displays labels for categorical data. This class extends `CategoryAxis` and implements `HorizontalAxis`.

12.34.2 Constructors

There are two constructors defined, one that sets up the axis with mostly default properties, and another that requires the caller to specify all the properties for the axis. Refer to the Javadoc or the source code for details.

> `public HorizontalCategoryAxis(String label);`
> Creates a new axis, using default values where necessary.

12.34.3 Attributes

The axis can display category labels with a horizontal or vertical orientation—this is controlled by the `VerticalCategoryLabels` attribute. The remaining properties for this class are inherited from `CategoryAxis`.

12.34.4 Notes

In the current implementation, this class can be used with `LinePlot` and `VerticalBarPlot`.

This class relies on the `Plot` to implement the `CategoryPlot` interface. This is because the axis has no control over the visual presentation of the data—in particular, the axis cannot know how the categories are to be distributed along the axis, so it must query the `Plot` via the defined interface.

See Also

`CategoryAxis`, `VerticalCategoryAxis`.

12.35 HorizontalCategoryPlot

12.35.1 Overview

This plot draws a chart using data from a `CategoryDataset`, where the categories are plotted against the vertical axis and the numerical data is plotted against the horizontal axis.

12.35.2 Constructors

This class provides two constructors—one that requires all the attributes for the plot to be specified, the other assumes a number of default values. Refer to the Javadoc or the source code for details.

12.35.3 Methods

Some notes on the methods for `HorizontalCategoryPlot`:

> `public double getCategoryCoordinate(...);`
> This method returns the y-coordinate of the center of the specified category. The category axis will call this method to determine where to place the category labels, because it has no knowledge of the distribution of categories (these could vary, depending on the nature of the plot).

12.35.4 Notes

This class inherits most of its functions from the `CategoryPlot` class.

See Also

`CategoryPlot`, `HorizontalValuePlot`, `VerticalCategoryPlot`.

12.36 HorizontalDateAxis

12.36.1 Overview

An axis that displays numerical data in *date format*—this class extends `DateAxis` and implements `HorizontalAxis`.

12.36.2 Attributes

The axis can display category labels with a horizontal or vertical orientation—this is controlled by the `verticalTickLabels` property.

The remaining properties for this class are inherited from `DateAxis`. Although the axis displays dates for tick labels, it is still working with `Number` objects. The numbers are interpreted as the number of milliseconds since 1 January 1970 (that is, the encoding used by `java.util.Date`).

See Also

`DateAxis`.

12.37 HorizontalIntervalBarRenderer

12.37.1 Overview

A *category item renderer* that draws horizontal bars representing an interval. This renderer requires data from the `IntervalCategoryDataset`.

See Also

`HorizontalCategoryPlot`, `CategoryItemRenderer`.

12.38 HorizontalNumberAxis

12.38.1 Overview

An horizontal axis that displays numerical data—this class extends `NumberAxis` and implements `HorizontalAxis`.

12.38.2 Constructors

There are three constructors for this class. One requires the caller to specify all the axis properties, while the other two use some default properties. Refer to the Javadoc or the source code for details.

12.38.3 Methods

Some notes on the methods in `HorizontalNumberAxis`:

`public void autoAdjustRange();`
Obtains the minimum and maximum data values from the `Plot`, provided that it implements `HorizontalValueRange`, and adjusts the axis range accordingly. Note that the `autoRangeIncludesZero` flag is checked in this method.

`public void refreshTicks(...);`
A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the `autoTickUnits` flag, and automatically determines a suitable "standard" tick size if required.

12.38.4 Notes

Refer to the Javadoc HTML files and the source code for details.

See Also
`NumberAxis`, `HorizontalNumberAxis`.

12.39 HorizontalNumberAxis3D

12.39.1 Overview

A horizontal number axis that works with the horizontal 3D bar chart.

12.40 HorizontalSymbolicAxis

12.40.1 Overview

An axis that displays numerical data using symbols.

See Also
`HorizontalNumberAxis`.

12.41 HorizontalValuePlot

12.41.1 Overview

An interface that returns the minimum and maximum values in the "horizontal direction" for a two-dimensional plot. The values could be from the dataset's domain or range, depending on the orientation of the plot.

This interface is known to be implemented by `HorizontalBarPlot`.

12.41.2 Methods

This interface has two methods:

`public Number getMinimumHorizontalDataValue();`
Returns the minimum data value in the horizontal direction for the plot;

`public Number getMaximumHorizontalDataValue();`
Returns the maximum data value in the horizontal direction for the plot;

12.41.3 Notes

Refer to the Javadoc HTML files and source code for details.

See Also
`VerticalValuePlot`.

12.42 ImageTitle

12.42.1 Overview

A chart title that displays an image.

12.42.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`AbstractTitle`.

12.43 JFreeChart

12.43.1 Overview

The `JFreeChart` class controls the entire chart generation process. It co-ordinates a collection of other classes with the aim of rendering charts that look good at arbitrary sizes.

JFreeChart has been designed to draw charts onto a Java 2D graphics device (`java.awt.Graphics2D`) which means that charts can be drawn on any device supported by Java. Usually, developers are interested in drawing charts on the screen, but you have the option to also output charts to the printer, an offscreen image buffer, a scalable vector graphics (SVG) generator, a PDF generator or whatever. Thanks to `Graphics2D` the same drawing code is used in all cases.

12.43.2 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.

12.43.3 Attributes

The `JFreeChart` class has the following attributes:

Attribute:	**Description:**
`titles`	A list of the titles for the chart.
`legend`	The chart legend.
`plot`	The plot.
`antialias`	A flag that indicates whether or not the chart should be drawn with anti-aliasing.
`backgroundPaint`	The background paint for the chart.
`backgroundImage`	An optional background image for the chart.
`backgroundImageAlpha`	The alpha transparency for the background image.

12.43.4 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).

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

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

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

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

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

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

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

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

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

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

12.43.5 Notes

The `ChartFactory` class provides some convenient 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 Java2 improves).

A chart can have multiple titles (see `AbstractTitle`), although often you will require just one title or no title at all.

See Also

`ChartFactory`, `ChartPanel`, `Plot`.

12.44 JFreeChartConstants

12.44.1 Overview

A collection of constants used by the JFreeChart class.

See Also

`JFreeChart`.

12.45 JFreeChartInfo

12.45.1 Overview

Information about the JFreeChart class library.

See Also

`JFreeChart`.

12.46 Legend

12.46.1 Overview

The base class for a chart legend.

12.46.2 Notes

This class implements a listener mechanism which can be used by subclasses. Refer to Javadoc HTML files and source code for details.

See Also

`StandardLegend`.

12.47 LegendItem

12.47.1 Overview

An item within a legend.

See Also

`Legend`.

12.48 LegendItemCollection

12.48.1 Overview

A collection of legend items.

See Also

`Legend`.

12.49 LegendItemLayout

12.49.1 Overview

An interface for laying out a collection of legend items.

12.49.2 Notes

This code is incomplete.

See Also

`Legend`.

12.50 LineAndShapeRenderer

12.50.1 Overview

A renderer used by the `HorizontalCategoryPlot` class to draw line plots with a `CategoryDataset`. This renderer can represent data values using shapes, lines or shapes and lines.

12.50.2 Constructors

The default constructor creates a renderer that draws both shapes and lines:

> `public LineAndShapeRenderer();`
> Creates a new `LineAndShapeRenderer` that draws both shapes and lines.

The other constructor allows you to specify the type of renderer:

> `public LineAndShapeRenderer(int type);`
> Creates a new `LineAndShapeRenderer` of the specified type. Use one of the constants defined by this class: `SHAPES`, `LINES`, or `SHAPES_AND_LINES`.

12.50.3 Methods

This class implements the `drawCategoryItem(...)` method that is defined in the `CategoryItemRenderer` interface.

12.50.4 Notes

The renderer is *immutable*, meaning that once an instance is created its properties cannot be changed. This property is relied upon to ensure that the appearance of a plot can be changed only in ways that are known to the plot (so that the plot can notify registered listeners that it has changed).

Refer to Javadoc HTML files and source code for further details.

See Also

`CategoryItemRenderer`.

12.51 Marker

12.51.1 Overview

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

See Also

`CategoryPlot`, `XYPlot`.

12.52 MeterLegend

12.52.1 Overview

To be documented.

12.53 MeterPlot

12.53.1 Overview

A plot that displays a single value in a meter (or gauge).

12.53.2 Notes

This was contributed by Hari.

See Also

`Plot`.

12.54 NumberAxis

12.54.1 Overview

The base class for axes (both horizontal and vertical) that display numerical data—extends `ValueAxis`.

12.54.2 Constructors

The `NumberAxis` class is `abstract`. Therefore you cannot instantiate this class directly—you must use a subclass (for example, `HorizontalNumberAxis` or `VerticalNumberAxis`).

Subclasses can call one of two constructors for the `NumberAxis` class. The simpler version requires only the axis label to be specified, with all other attributes taking default values:

```
protected NumberAxis(String label);
```
Creates a new `NumberAxis`.

The other constructor takes an extensive list of parameters, allowing much greater control over the construction of the axis. Refer to the Javadoc HTML pages or the source code for details.

12.54.3 Attributes

The following table lists the properties defined by `NumberAxis`:[8]

[8] Keep in mind that many other attributes are inherited from `ValueAxis`.

Attribute:	Description:
`MinimumAxisValue`	The lowest value displayed on the axis.
`MaximumAxisValue`	The highest value displayed on the axis.
`AutoRangeIncludesZero`	A flag that indicates whether or not zero is always included when the axis range is determined automatically.
`AutoRangeMinimumSize`	If the axis range is determined automatically, it is guaranteed never to be less that this value.
`UpperMargin`	The margin to allow at the upper end of the axis scale (expressed as a percentage of the total axis range).
`LowerMargin`	The margin to allow at the lower end of the axis scale (expressed as a percentage of the total axis range).
`TickUnit`	The spacing between ticks on the axis.
`StandardTickUnits`	A collection of standard tick units. If *auto-tick-selection* is on, one of these tick units will be selected automatically.

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_UPPER_MARGIN`	`0.05` (5 percent)
`DEFAULT_LOWER_MARGIN`	`0.05` (5 percent)
`DEFAULT_MINIMUM_AUTO_RANGE`	`new Double(0.0000001);`
`DEFAULT_TICK_UNIT`	`new NumberTickUnit(new Double(1.0), new DecimalFormat("0"));`

12.54.4 Methods

To set the lower bound for the axis:

```
public void setMinimumAxisValue(double value);
```

Sets the lower bound for the axis. If the `AutoRange` attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

To set the upper bound for the axis:

```
public void setMaximumAxisValue(double value);
```

Sets the upper bound for the axis. If the `AutoRange` attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

If you have set the `AutoRange` flag to `true` (so that the axis range automatically adjusts to fit the current data), you may also want to set the `AutoRangeIncludes-Zero` flag to ensure that the axis range always includes zero:

```
public void setAutoRangeIncludesZero(boolean flag);
```

Sets the `AutoRangeIncludesZero` flag.

When the `AutoTickUnit` property 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.

12.54.5 Notes

This class defines a default set of standard tick units. You can override the default settings by calling the `setStandardTickUnits(...)` method.

See Also

`TickUnits`, `ValueAxis`.

12.55 NumberTickUnit

12.55.1 Overview

A numerical tick unit. The `NumberAxis` class creates a collection of standard tick units from which it can choose an appropriate tick unit for the range of data it is trying to display.

12.55.2 Constructors

The standard constructor:

> `public NumberTickUnit(Number value, NumberFormat formatter);`
> Creates a new number tick unit.

12.55.3 Notes

Extends the `TickUnit` class.

See Also

`TickUnit`.

12.56 OverlaidVerticalCategoryPlot

12.56.1 Overview

A recent addition to the JFreeChart class library that draws overlaid vertical category plots. To be documented.

12.57 OverlaidXYPlot

12.57.1 Overview

A subclass of `XYPlot`, this class allows you to combine multiple subplots within a single chart. As far as possible, this class tries to behave in exactly the same way as a regular `XYPlot`. Setting axis ranges, background colors and so forth should be no different to usual.

One important difference between this class and `XYPlot` is that you do not supply a dataset for overlaid plots. Each of the subplots maintains its own dataset.

All the subplots (instances of `XYPlot`) should have `null` axes, because they share the axes managed by the `OverlaidXYPlot`. When you set the properties of an axis belonging to an overlaid plot (the *parent plot*) all of the subplots will update to reflect the change.

Figure 6 illustrates the relationship between the parent plot and its subplots.

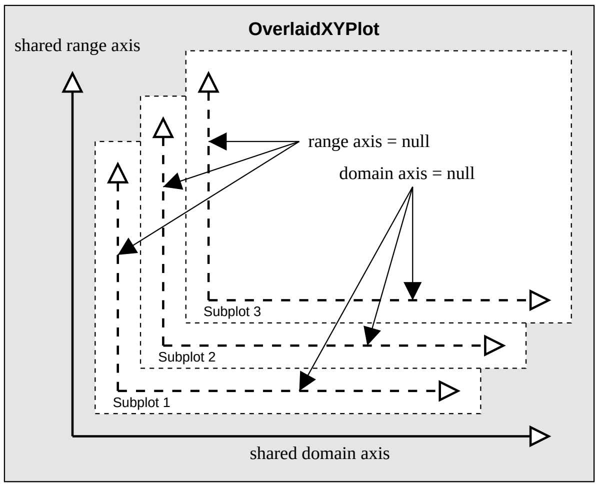


Figure 6: OverlaidXYPlot axes

12.57.2 Constructors

To construct a new `OverlaidXYPlot`:

`public OverlaidXYPlot(ValueAxis domain, ValueAxis range);`
Creates a new plot with the specified axes. No dataset is necessary, since the subplots (which you must add) maintain their own datasets.

Another constructor, which takes a domain axis label and a range axis label as arguments, creates a new plot with numerical axes. This is provided for convenience, allowing you to construct a new plot without having to first construct axes.

12.57.3 Methods

To add a subplot:

`public void add(XYPlot subplot);`
Adds a subplot. The subplot can be almost any instance of `XYPlot` and should have both its axes set to `null`.

12.57.4 Notes

The dataset for this class should be set to `null`.

The subplots managed by this class should have their domain and range axes set to `null`.

A demonstration of this type of plot is described in section 8.2.

See Also

`XYPlot`, `CombinedXYPlot`.

12.58 PeriodMarkerPlot

12.58.1 Overview

A plot that highlights time periods using different colors.

12.58.2 Notes

This was contributed by Sylvain Vieujot. I haven't done any work with this class yet, but my initial thought is that it could be converted to an `XYItemRenderer`.

See Also

`XYPlot`.

12.59 PiePlot

12.59.1 Overview

The `PiePlot` class draws pie charts, using data obtained through the `PieDataset` interface (part of the JCommon Class Library).

12.59.2 Constructors

The default constructor:

> `protected PiePlot();`
> Creates a pie plot with default attributes.

12.59.3 Attributes

The `PiePlot` class has the following attributes:

Attribute:	Description:
`InteriorGapPercent`	The space to leave blank around the outside of the pie.
`Circular`	Circular or elliptical pie.
`RadiusPercent`	Controls the radius of the unexploded pie.
`SectionLabelType`	The type of labels for the pie sections.
`SectionLabelFont`	The font for the section labels.
`SectionLabelPaint`	The color for the section labels.
`SectionLabelGapPercent`	The gap for the section labels.
`ExplodePercentages[]`	The amount to 'explode' each pie section.
`PercentFormatter`	A formatter for the percentage labels.
`ToolTipGenerator`	A plug-in tooltip generator.

The following default values are used where necessary:

Name:	Value:
`DEFAULT_INTERIOR_GAP`	`0.20` (20 percent)
`DEFAULT_RADIUS`	`1.00` (100 percent)
`DEFAULT_SECTION_LABEL_FONT`	`new Font("SansSerif", Font.PLAIN, 10);`
`DEFAULT_SECTION_LABEL_PAINT`	`Color.black;`
`DEFAULT_SECTION_LABEL_GAP`	`0.10` (10 percent)

12.59.4 Methods

A pie plot is drawn with this method:

> `public void draw(Graphics2D g2, Rectangle2D drawArea, ToolTips tooltips);`
> Draws the pie plot within the specified drawing area.
>
> If `tooltips` is not `null`, then tooltip regions will be recorded for each pie section as the pie plot is drawn. This information can be used later to display tooltips.

The `JFreeChart` class usually calls the `draw(...)` method for you.

You can control the style of the labels for each section of the pie chart:

> `public void setSectionLabelType(int type);`
> Sets the type of label to display next to each section of the pie chart. Use one of the following constants: `NO_LABELS`, `NAME_LABELS`, `VALUE_LABELS`, `PERCENT_LABELS`, `NAME_AND_VALUE_LABELS`, `NAME_AND_PERCENT_LABELS` and `VALUE_AND_PERCENT_LABELS`.

To set the tooltip generator (optional) for the pie plot:

> `public void setToolTipGenerator(PieToolTipGenerator generator) ;`
> Registers a tooltip generator with the pie plot. If you write your own generator, you can have full control over the tooltip text that is generated for each pie section.

12.59.5 Notes

`PiePlot` inherits axes from the `Plot` class. You should leave these set to `null`.

See Also

`Plot`.

12.60 Plot

12.60.1 Overview

An abstract base class that controls the visual representation of data in a chart.

The `JFreeChart` class maintains a reference to one `Plot`. The plot, in turn, manages the `Dataset` and the axes (if there are any).

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.

Figure 7 illustrates the plot class hierarchy.

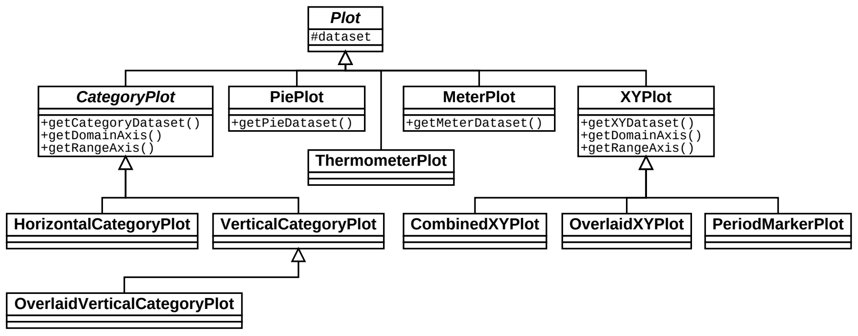


Figure 7: Plot classes

12.60.2 Constructors

This class is abstract, so the constructors are `protected`.
The first constructor accepts a `Dataset` and uses default values for all other attributes:

`protected Plot(Dataset data);`

Creates a new `Plot`. The plot registers with the dataset to receive notification of any changes.

The other constructor allows you to supply custom values for most attributes.

12.60.3 Attributes

The `Plot` class has the following attributes:

Attribute:	**Description:**
`Dataset`	The dataset.
`Insets`	The amount of space to leave around the outside of the plot.
`BackgroundPaint`	The color used to draw the background of the plot area.
`BackgroundImage`	An optional background image.
`BackgroundAlpha`	The alpha transparency used to draw the background.
`OutlineStroke`	The pen/brush used to draw an outline around the plot area.
`OutlinePaint`	The color used to draw an outline around the plot area.
`ForegroundAlpha`	The alpha transparency used to draw the foreground.
`SeriesPaint`	An array of `Paint` objects used for the series colors.
`SeriesStroke`	An array of `Stroke` objects used for drawing series.
`SeriesOutlineStroke`	An array of `Stroke` objects used for drawing series.
`SeriesOutlinePaint`	An array of `Paint` objects used for the series outline colors.

12.60.4 Methods

The most important method is the `draw(...)` method:

`public abstract void draw(Graphics2D g2, Rectangle2D plotArea, ChartRenderingInfo info);`

Draws the chart using the supplied `Graphics2D`. The plot should be confined to the specified 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`.

Note that the `draw(...)` method is called by the `JFreeChart` class. You don't normally need to call it yourself.

12.60.5 Notes

The `Plot` class works with the `Dataset` interface (and its extensions) that are defined in the JCommon Class Library. You can download JCommon from:

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

See Also

JFreeChart, CategoryPlot, XYPlot.

12.61 PlotException

12.61.1 Overview

A general purpose exception that can be generated by subclasses of `Plot`.

12.61.2 Notes

At the current time, there isn't any code that throws this type of exception, but the class is being retained for future use.

12.62 PlotNotCompatibleException

12.62.1 Overview

An exception that indicates that an attempt has been made to assign a plot to a chart where the plot is not compatible with the chart's current `Dataset`. For example, an `XYPlot` will not work with a `CategoryDataset`.

12.62.2 Constructors

To create a new exception:

```
public AxisNotCompatibleException(String message);
```

Creates a new exception.

12.62.3 Notes

The `PlotNotCompatibleException` class is a subclass of `RuntimeException`.

See Also

`AxisNotCompatibleException`.

12.63 SeriesShapeFactory

12.63.1 Overview

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

12.64 ShapeFactory

12.64.1 Overview

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

12.65 SignalRenderer

12.65.1 Overview

A plot that draws different signals depending on the direction of the data.

12.65.2 Notes

This was contributed by Sylvain Vieujot.

See Also
`Plot`.

12.66 Spacer

12.66.1 Overview

A class that is used to specify spacing information within charts.

12.66.2 Notes

This class is intended to replace the use of `Insets`.

See Also
`Plot`.

12.67 StackedHorizontalBarRenderer

12.67.1 Overview

A renderer for the `HorizontalBarPlot` class that draws stacked bars.

12.67.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`HorizontalBarPlot`, `HorizontalBarRenderer`.

12.68 StackedVerticalBarRenderer

12.68.1 Overview

A renderer for the `VerticalBarPlot` class that draws stacked bars.

12.68.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`VerticalBarPlot`.

12.69 StackedVerticalBarRenderer3D

12.69.1 Overview

A renderer for the `VerticalBarPlot` class that draws stacked bars with a 3D-effect.

12.69.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`VerticalBarPlot`.

12.70 StandardLegend

12.70.1 Overview

This class is soon to be replaced by `LegendTitle`.

12.71 StandardXYItemRenderer

12.71.1 Overview

The default renderer for the `XYPlot` class. This renderer represents data by drawing lines between *(x, y)* data points and/or drawing shapes at each *(x, y)* data point.

12.71.2 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`.

12.71.3 Notes

This class implements the `XYItemRenderer` interface.

The `XYPlot` class will use an instance of this class as its default renderer.

See Also

`XYPlot`, `XYItemRenderer`.

12.72 TextTitle

12.72.1 Overview

A standard chart title—extends `AbstractTitle`.

12.72.2 Notes

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

See Also

`AbstractTitle`.

12.73 Tick

12.73.1 Overview

A utility class representing a tick on an axis. Used temporarily during the drawing process only.

12.73.2 Constructors

The standard constructor:

```
public Tick(Object value, String text, float x, float y)
```
Creates a tick.

See Also

`TickUnit`.

12.74 TickUnit

12.74.1 Overview

An abstract class representing a tick unit. Subclasses include `NumberTickUnit`.

12.74.2 Constructors

The standard constructor:

```
public TickUnit(Number value);
```
Creates a new tick value.

12.74.3 Notes

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

See Also
`NumberTickUnit`.

12.75 TickUnits

12.75.1 Overview

A collection of tick units. Used by the `Number` axis class to store a list of "standard" tick units, from which an appropriate tick unit is selected as the chart is being redrawn.

12.75.2 Constructors

The default constructor:

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

12.75.3 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 closest in size to another tick unit:

```
public TickUnit getNearestTickUnit(TickUnit unit);
```
Returns the tick unit that is closest in size to the specified unit.

12.75.4 Notes

The `NumberAxis` class has a private method `createStandardTickUnits()` that generates a tick unit collection (of standard tick sizes) for use by numerical axes.

See Also
`TickUnit`.

12.76 ValueAxis

12.76.1 Overview

The base class for all (horizontal and vertical) axes that display "values". Ultimately, values are represented as `double` primitives, but subclasses of `ValueAxis` have been implemented that give the appearance of working with `Number` and `Date` objects.

Known subclasses of `ValueAxis` include `DateAxis` and `NumberAxis`.

12.76.2 Constructors

To construct a `ValueAxis`:

> `protected ValueAxis(String label);`
> Creates a `ValueAxis` with the specified label. All other attributes take default values.

If you want more control over the settings for the axis, there is another constructor that takes a full range of arguments specifying the atttributes for the axis. Refer to the Javadoc HTML files or the source code for details.

12.76.3 Attributes

The `ValueAxis` class has the following attributes:

Attribute:	**Description:**
`AutoRange`	A flag controlling whether or not the axis automatically adjusts its range to reflect the range of data values.
`AutoTickUnitSelection`	A flag controlling whether or not the tick units are selected automatically.
`CrosshairVisible`	A flag controlling whether or not the crosshair is visible.
`CrosshairValue`	The value at which the crosshair is drawn.
`CrosshairStroke`	The pen/brush used to draw the crosshair.
`CrosshairPaint`	The paint used to draw the crosshair.
`CrosshairLockedOnData`	A flag controlling whether or not the crosshair is locked to a data point.
`GridLinesVisible`	A flag controlling whether or not grid lines are displayed.
`GridLineStroke`	The *stroke* used to draw the grid lines.
`GridLinePaint`	The color for the grid lines.

The following default values are used for attributes wherever necessary:

Name:	**Value:**
`DEFAULT_AUTO_RANGE`	`true;`
`DEFAULT_MINIMUM_AXIS_VALUE`	`0.0;`
`DEFAULT_MAXIMUM_AXISVALUE`	`1.0;`
`DEFAULT_GRID_LINE_STROKE`	`new BasicStroke(0.5f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL, 0.0f, new float[] 2.0f, 2.0f, 0.0f);`
`DEFAULT_GRID_LINE_PAINT`	`Color.grey;`

12.76.4 Methods

A key function for a `ValueAxis` is to convert a data value to an output coordinate for plotting purposes. The output coordinate will be dependent on the area into which the data is being drawn:

> `public double translateValueToJava2D(double value, Rectangle2D dataArea);`
> Converts a data value into a co-ordinate within the `dataArea`. The `dataArea` is the rectangle inside the plot's axes.

See Also

`Axis`, `DateAxis`, `NumberAxis`.

12.77 VerticalAxis

12.77.1 Overview

An interface that *must* be implemented by all vertical axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

12.77.2 Methods

The interface defines two methods. The plot chooses which of these two methods to call when laying out the axes.

> **`public Rectangle2D reserveAxisArea(Graphics2D g2, Plot plot, Rectangle2D drawArea, double reservedHeight);`**
> Calculates the area that the vertical axis requires to draw itself. If this method is used, it will be called after the horizontal axis has estimated the height that it requires—the argument **`reservedHeight`** contains this value.

> **`public double reserveWidth(Graphics2D g2, Plot plot, Rectangle2D drawArea);`**
> Estimates the width that the vertical axis requires to draw itself. If this method is used, it will be called before the horizontal axis is asked to calculate the area that it requires—the width returned by this method will be passed to the horizontal axis.

See Also

`HorizontalAxis`.

12.78 VerticalBarRenderer

12.78.1 Overview

A renderer that draws ordinary bars on a `VerticalBarPlot`.

12.78.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`VerticalCategoryPlot`.

12.79 VerticalBarRenderer3D

12.79.1 Overview

A specialised renderer for the `VerticalBarPlot` class that draws the bars with a 3D-effect.

12.79.2 Notes

This class was contributed by Serge V. Grachov.

Refer to Javadoc HTML files and source code for details.

See Also

`VerticalCategoryPlot`.

12.80 VerticalCategoryAxis

12.80.1 Overview

A vertical axis that displays categorical data. This class extends `CategoryAxis`. As for the other category axes, this class relies on the plot to provide information about how the categories are distributed along the axis (this information is obtained via the `CategoryPlot` interface).

12.80.2 Constructors

There are two constructors for this class. One requires all the attributes for the axis to be specified, the other provides for default values on some attributes.

12.81.2 Constructors

The simplest constructor requires only the axes to be specified:

> `public VerticalCategoryPlot(CategoryAxis domainAxis, ValueAxis rangeAxis);`
> Creates a vertical category plot. Default values are assumed for most attributes.

For more complete control, use the following constructor:

> `public VerticalCategoryPlot(CategoryAxis domainAxis, ValueAxis rangeAxis, Insets insets, double introGapPercent, double trailGapPercent, double categoryGapPercent, double itemGapPercent, CategoryToolTipGenerator toolTipGenerator);`
> Creates a vertical category plot.

12.81.3 Methods

The category axis will need to ask the plot for the coordinate of a particular category, since the plot controls the distribution of the categories. This method is used:

> `public double getCategoryCoordinate(...);`
> This method returns the x-coordinate of the center of the specified category. The category axis will call this method to determine where to place the category labels, because it has no knowledge of the distribution of categories (these could vary depending on the nature of the plot).

12.81.4 Notes

The bar widths cannot be controlled directly. Instead, you set the amount (percentage) of the total space that should be allocated to the gaps *between* the bars, and then the bar widths are determined automatically.

See Also

`CategoryPlot`, `HorizontalCategoryPlot`, `VerticalValuePlot`.

12.82 VerticalIntervalBarRenderer

12.82.1 Overview

To be documented.

12.83 VerticalLogarithmicAxis

12.83.1 Overview

A numerical axis that displays values using a logarithmic scale.

12.83.2 Notes

An equivalent class `HorizontalLogarithmicAxis` has now been implemented.

See Also

`NumberAxis`.

12.84 VerticalNumberAxis

12.84.1 Overview

A vertical axis that displays numerical data—this class extends `NumberAxis`.

12.84.2 Constructors

There are three constructors for this class. One requires all the attributes for the axis to be specified, the other two provide for default values on some attributes. Refer to the Javadoc or source code for details.

12.84.3 Methods

A list of important methods:

> `public void autoAdjustRange();`
> This method obtains the maximum and minimum data values from the `Plot`, provided that it implements `VerticalValueRange`, and adjusts the axis range accordingly. Note that the `autoRangeIncludesZero` flag is checked in this method.
>
> `public void refreshTicks(...);`
> A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the `autoTickUnits` flag, and automatically determines a suitable "standard" tick size if required.
>
> `private void calculateAutoTickUnits(...);`
> This method is used to pick a standard tick size from the array defined in `NumberAxis`. The approach used is to find the smallest tick units such that the tick labels do not overlap.

See Also

`NumberAxis`, `HorizontalNumberAxis`.

12.85 VerticalNumberAxis3D

12.85.1 Overview

A vertical axis that draws itself with a 3D-effect. In all other respects, the axis should behave in the same way as the `VerticalNumberAxis` class.

12.85.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`VerticalNumberAxis`.

12.86 VerticalSymbolicAxis

12.86.1 Overview

A numerical axis that displays values using symbols.

See Also

`NumberAxis`.

12.87 VerticalValuePlot

12.87.1 Overview

An interface that returns minimum and maximum data values in the "vertical direction" for a two-dimensional plot. The values could be from the dataset domain or range, depending on the orientation of the plot.

12.87.2 Methods

This interface has two methods:

```
public Number getMinimumVerticalDataValue();
```
Returns the minimum data value in the vertical direction for the plot.

```
public Number getMaximumVerticalDataValue();
```
Returns the maximum data value in the vertical direction for the plot.

12.87.3 Notes

This interface is known to be implemented by `VerticalCategoryPlot` and `XYPlot`.

See Also

`HorizontalValuePlot`.

12.88 VerticalXYBarRenderer

12.88.1 Overview

Represents data from an `IntervalXYDataset` in the form of vertical bars on an `XYPlot`.

12.88.2 Constructors

The only constructor takes no arguments.

12.88.3 Methods

The `drawItem(...)` method handles the rendering of a single item for the plot.

12.88.4 Notes

This renderer casts the dataset to `IntervalXYDataset`, so you should ensure that the plot is supplied with the correct type of data. Refer to Javadoc HTML files and source code for further details.

See Also

`XYPlot`.

12.89 WindItemRenderer

12.89.1 Overview

A renderer that `XYPlot` uses to draw wind plots.

See Also

`XYPlot`.

12.90 XYItemRenderer

12.90.1 Overview

An interface that must be implemented by a *renderer* so that it can work with an `XYPlot`. By changing the renderer for an `XYPlot`, you can change the appearance of the data items within the plot.

Figure 8 illustrates the hierarchy of classes that implement this interface.

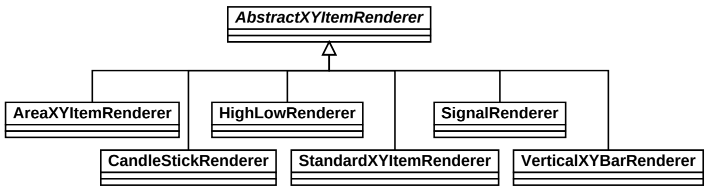


Figure 8: Plot classes

12.90.2 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.

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`.

12.90.3 Notes

Some renderers require a specific extension of `XYDataset`. For example, the `HighLowRenderer` requires a `HighLowDataset`.

See Also

`XYPlot`.

12.91 XYPlot

12.91.1 Overview

Draws a visual representation of data from an `XYDataset`, where the horizontal axis measures the x-values and the vertical axis measures the y-values.

It is possible to display time series data with `XYPlot` by employing a `HorizontalDateAxis` in place of the usual `HorizontalNumberAxis`. In this case, the x-values are interpreted as milliseconds as used in `java.util.Date`.

12.91.2 Constructors

The simplist constructor requires just the axes to be specified:

```
public XYPlot(ValueAxis horizontalAxis, ValueAxis verticalAxis);
```

Creates an XY plot. Default values are used where necessary.

```
public XYPlot(ValueAxis horizontalAxis, ValueAxis verticalAxis,
Insets insets, Paint background, Stroke outlineStroke, Paint outlinePaint);
```

Creates an XY plot.

12.91.3 Methods

To get the current renderer for the plot:

```
public XYItemRenderer getItemRenderer();
```

Returns the current renderer.

To set a new renderer for the plot:

```
public void setItemRenderer(XYItemRenderer renderer);
```

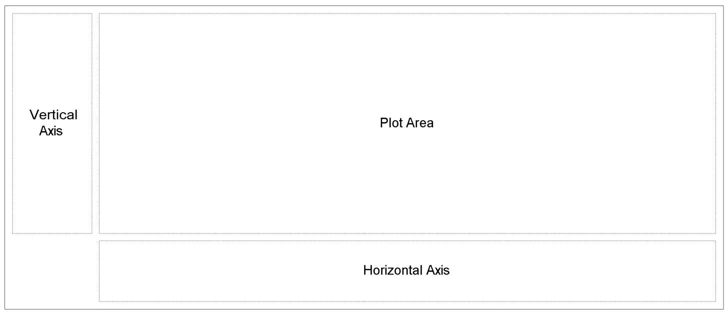
Sets a new renderer.

12.91.4 Notes

The `XYPlot` class works with a *renderer* to control the visual representation of the data. By default, a renderer is installed that draws lines between each of the data points.

`XYPlot` implements both `HorizontalValuePlot` and `VerticalValuePlot`, enabling the axes to automatically determine the range of data that is available for the plot.

Axes are laid out at the left and bottom of the drawing area. The space allocated for the axes is determined automatically. The following diagram shows how this area is divided:



Determining the dimensions of these regions is an awkward problem. The plot area can be resized arbitrarily, but the vertical axis and horizontal axis sizes are more difficult. Note that the height of the vertical axis is related to the height of the horizontal axis, and, likewise, the width of the vertical axis is related to the width of the horizontal axis. This results in a "chicken and egg" problem, because changing the width of an axis can affect its height (especially if the tick units change with the resize) and changing its height can affect the width (for the same reason).

See Also

`Plot`, `OverlaidXYPlot`, `XYItemRenderer`.

12.92 XYStepRenderer

12.92.1 Overview

To be documented.

13 Package: com.jrefinery.chart.data

13.1 Introduction

This package contains some classes for data fitting. These will eventually be rewritten and moved into another package.

13.2 LinearPlotFitAlgorithm

13.2.1 Overview

Not yet documented.

13.2.2 Notes

Refer to Javadoc HTML files and source code for details.

13.3 MovingAveragePlotFitAlgorithm

13.3.1 Overview

Not yet documented.

13.3.2 Notes

Refer to Javadoc HTML files and source code for details.

13.4 PlotFit

13.4.1 Overview

Not yet documented.

13.4.2 Notes

Refer to Javadoc HTML files and source code for details.

13.5 PlotFitAlgorithm

13.5.1 Overview

Not yet documented.

13.5.2 Notes

Refer to Javadoc HTML files and source code for details.

14 Package: com.jrefinery.chart.entity

14.1 Introduction

The `com.jrefinery.chart.entity` package contains classes that represent entities in a chart.

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 structure. 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 and returning detailed information for chart mouse events.

14.2 CategoryItemEntity

14.2.1 Overview

This class is used to convey information about an item within a category plot. The information captured includes the area occupied by the item, the tool tip text generated for the item, and the series and category that the item represents.

14.2.2 Constructors

To construct a new instance:

```
public CategoryItemEntity(Shape area, String toolTipText, int series,
Object category);
```

Creates a new entity instance.

14.2.3 Methods

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

14.2.4 Notes

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

See Also

`ChartEntity`, `CategoryPlot`.

14.3 ChartEntity

14.3.1 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.

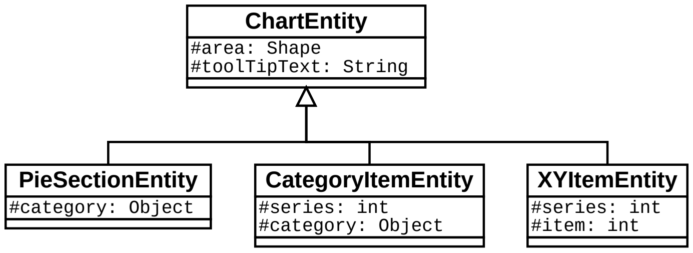


Figure 9: Chart entity classes

14.3.2 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.

14.3.3 Methods

Accessor methods are implemented for the `area` and `toolTipText` attributes.

14.3.4 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`.

14.4 EntityCollection

14.4.1 Overview

An interface that defines the API for a *chart entity collection*. The `Chart-RenderingInfo` class uses a chart entity collection to record where items have been drawn when a chart is rendered using a `Graphics2D` instance.

14.4.2 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.

14.4.3 Notes

The `StandardEntityCollection` class provides a basic implementation of this interface.

See Also

`ChartEntity`, `StandardEntityCollection`.

14.5 PieSectionEntity

14.5.1 Overview

This class is used to convey information about an item within a pie plot. The information captured includes the area occupied by the item, the tool tip text generated for the item and the category that the item represents.

14.5.2 Constructors

To construct a new instance:

```
public PieSectionEntity(Shape area, String toolTipText, Object category);
```
Creates a new entity object.

14.5.3 Methods

Accessor methods are implemented for the `category` attribute. Other methods are inherited from the `ChartEntity` class.

14.5.4 Notes

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

See Also

`ChartEntity`, `PiePlot`.

14.6 StandardEntityCollection

14.6.1 Overview

A basic implementation of the `EntityCollection` interface. This class is used to store a collection of chart entity objects from one rendering of a chart (see the `ChartRenderingInfo` class for more details).

14.6.2 Methods

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

14.6.3 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.[9]

See Also

`ChartEntity`, `EntityCollection`.

14.7 XYItemEntity

14.7.1 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.

14.7.2 Constructors

To construct a new instance:

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

Creates a new entity object.

[9] 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.

14.7.3 Methods

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

14.7.4 Notes

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

See Also

`ChartEntity`, `XYPlot`.

15 Package: com.jrefinery.chart.event

15.1 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.

15.2 AxisChangeEvent

15.2.1 Overview

An event that is used to provide information about changes to axes.

See Also
`AxisChangeListener`.

15.3 AxisChangeListener

15.3.1 Overview

An interface through which axis change event notifications are posted. 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.

15.3.2 Methods

The interface defines a single method:

```
public void axisChanged(AxisChangeEvent event);
```
Receives notification of a change to an axis.

See Also
`AxisChangeEvent`.

15.4 ChartChangeEvent

15.4.1 Overview

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

See Also
`ChartChangeListener`.

15.5 ChartChangeListener

15.5.1 Overview

An interface through which chart change event notifications are posted. 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.

15.5.2 Methods

The interface defines a single method:

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

Receives notification of a change to a chart.

See Also

`ChartChangeEvent`.

15.6 LegendChangeEvent

15.6.1 Overview

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

See Also

`LegendChangeListener`.

15.7 LegendChangeListener

15.7.1 Overview

An interface through which legend change event notifications are posted. 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.

15.7.2 Methods

The interface defines a single method:

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

Receives notification of a change to a legend.

See Also

`LegendChangeEvent`.

15.8 PlotChangeEvent

15.8.1 Overview

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

See Also
`PlotChangeListener`.

15.9 PlotChangeListener

15.9.1 Overview

An interface through which plot change event notifications are posted. 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.

15.9.2 Methods

The interface defines a single method:

```
public void plotChanged(PlotChangeEvent event);
```

Receives notification of a change to a plot.

See Also
`PlotChangeEvent`.

15.10 TitleChangeEvent

15.10.1 Overview

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

See Also
`TitleChangeListener`.

15.11 TitleChangeListener

15.11.1 Overview

An interface through which title change event notifications are posted. 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.

15.11.2 Methods

The interface defines a single method:

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

Receives notification of a change to a title.

See Also
`TitleChangeEvent`.

16 Package: com.jrefinery.chart.tooltips

16.1 Introduction

This package contains some classes for generating tooltips.

16.2 CategoryToolTipGenerator

16.2.1 Overview

The interface that should be implemented by a *category tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a plot, and take full control over the tooltip text that is generated.

16.2.2 Methods

This interface defines a single method:

> `public String generateToolTip(CategoryDataset data, int series, Object category);`
> This method is called whenever the plot needs to generate a tooltip. It should return the tooltip text (which can be anything you want to make it).

16.2.3 Notes

The `StandardCategoryToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

16.3 PieToolTipGenerator

16.3.1 Overview

The interface that should be implemented by a *pie tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a `PiePlot`, and take full control over the tooltip text that is generated.

16.3.2 Methods

This interface defines a single method:

> `public String generateToolTip(PieDataset data, Object category);`
> This method is called whenever the `PiePlot` needs to generate a tooltip. It should return a `String` that will be used as the tooltip text.

16.3.3 Notes

The `StandardPieToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

See Also

`StandardPieToolTipGenerator`.

16.4 StandardCategoryToolTipGenerator

16.4.1 Overview

A default implementation of the `CategoryToolTipGenerator` interface.

16.4.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`CategoryToolTipGenerator`.

16.5 StandardHighLowToolTipGenerator

16.5.1 Overview

A default implementation of the `HighLowToolTipGenerator` interface.

16.5.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`HighLowToolTipGenerator`.

16.6 StandardPieToolTipGenerator

16.6.1 Overview

A default implementation of the `PieToolTipGenerator` interface.

16.6.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`PieToolTipGenerator`.

16.7 StandardToolTips

16.7.1 Overview

An implementation of the `ToolTips` interface, this class can be registered with a chart via the `setToolTips(...)` method and will collect tooltips as the chart is being drawn.

16.7.2 Constructors

Use the default constructor to create a new tooltip manager:

```
public StandardToolTips();
```

Creates a new tooltip manager.

16.7.3 Methods

This class provides implementations for all the methods in the `ToolTips` interface.

16.7.4 Notes

This implementation is not highly optimised. If you are using generating charts with large numbers of data items, you should either stop using tooltips, or write a more efficient implementation.

See Also

`ToolTips`.

16.8 StandardXYToolTipGenerator

16.8.1 Overview

A default implementation of the `XYToolTipGenerator` interface.

16.8.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`XYToolTipGenerator`.

16.9 ToolTip

16.9.1 Overview

A simple class representing a tooltip. It records the tooltip text, and the area that the tooltip applies to.

16.9.2 Notes

This class is immutable.

See Also

`ToolTipGenerator`.

16.10 ToolTipGenerator

16.10.1 Overview

Not yet documented.

16.10.2 Notes

Refer to Javadoc HTML files and source code for details.

16.11 ToolTips

16.11.1 Overview

An interface defining the methods to be supported by a *tooltip manager*.

If you set a tooltip manager for a chart, then it will collect tooltips as the chart is being drawn (provided that the `Plot` subclass is capable of generating tooltips). The `JFreeChartPanel` class makes use of this facility to provide chart tooltips.

16.11.2 Notes

By default, there is no tooltip manager set for a chart.

See Also

`StandardToolTips`.

16.12 XYToolTipGenerator

16.12.1 Overview

The interface that should be implemented by a *XY tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a plot, and take full control over the tooltip text that is generated.

16.12.2 Methods

This interface defines a single method:

```
public String generateToolTip(XYDataset data, int series, int item);
```

This method is called whenever the `XYPlot` needs to generate a tooltip. It should return a `String` that will be used as the tooltip text.

16.12.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also

`StandardXYToolTipGenerator`.

17 Package: com.jrefinery.chart.ui

17.1 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.

17.2 AxisPropertyEditPanel

17.2.1 Overview

Not yet documented.

17.2.2 Notes

Refer to Javadoc HTML files and source code for details.

17.3 ChartPropertyEditPanel

17.3.1 Overview

A panel that displays all the properties of a chart, and allows the user to edit the properties. The panel uses a `JTabbedPane` to display four sub-panels: a `TitlePropertyPanel`, a `LegendPropertyPanel`, a `PlotPropertyPanel` and a panel containing "other" properties (such as the anti-alias setting and the background paint for the chart).

The constructors for this class require a reference to a `Dialog` or a `Frame`. Whichever one is specified is passed on to the `TitlePropertyPanel` and is used if and when a sub-dialog is required for editing titles.

17.3.2 Notes

Refer to Javadoc HTML files and source code for details.

17.4 LegendPropertyEditPanel

17.4.1 Overview

Not yet documented.

17.4.2 Notes

Refer to Javadoc HTML files and source code for details.

17.5 NumberAxisPropertyEditPanel

17.5.1 Overview

Not yet documented.

17.5.2 Notes

Refer to Javadoc HTML files and source code for details.

17.6 PlotPropertyEditPanel

17.6.1 Overview

Not yet documented.

17.6.2 Notes

Refer to Javadoc HTML files and source code for details.

17.7 TitlePropertyEditPanel

17.7.1 Overview

Not yet documented.

17.7.2 Notes

Refer to Javadoc HTML files and source code for details.

18 Package: com.jrefinery.data

18.1 Introduction

This package is part of the JCommon Class Library, which can be downloaded from:

`http://www.jrefinery.com/jcommon/index.html`

The reference documentation for this package is included here, even though it is not strictly part of the JFreeChart Class Library, because JFreeChart makes extensive use of the interfaces and classes in this package.

18.2 AbstractDataset

18.2.1 Overview

A useful base class for implementing the `Dataset` interface (or extensions). This class provides a default implementation of the *change listener* mechanism.

18.2.2 Constructors

The default constructor:

`protected AbstractDataset();`
Allocates storage for the registered change listeners.

18.2.3 Methods

`public void addChangeListener(DatasetChangeListener listener);`
Registers a change listener with the dataset. The listener will be notified whenever the dataset changes.

`public void addChangeListener(DatasetChangeListener listener);`
Deregisters a change listener. The listener will be no longer be notified whenever the dataset changes.

18.2.4 Notes

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.

See Also

`Dataset`, `DatasetChangeListener`, `AbstractSeriesDataset`.

18.3 AbstractSeriesDataset

18.3.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

18.3.2 Constructors

The default constructor:

```
protected AbstractSeriesDataset();
```
Simply calls the constructor of the superclass.

18.3.3 Methods

Implementations are provided for the following methods:

```
public String[] getLegendItemLabels();
```
Returns an array of series names.

18.3.4 Notes

You can implement a dataset without subclassing `AbstractSeriesDataset`. This class is provided simply for convenience.

See Also

`Dataset`.

18.4 BasicTimeSeries

18.4.1 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 `TimePeriod`. Subclasses include `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`. Different subclasses of `TimePeriod` cannot be mixed in one time series.

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.

Here is an example showing how to create a series with quarterly data:

```
BasicTimeSeries series = new BasicTimeSeries("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 `BasicTimeSeries` objects can be added to a `TimeSeriesCollection` and used as the dataset for a chart in JFreeChart.

18.4.2 Constructors

To create a named time series containing no data:

> `public BasicTimeSeries(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 BasicTimeSeries(String name, Class timePeriodClass);`
> Creates an empty time series. The caller specifies the time period by specifying the class of the `TimePeriod` subclass (for example, `Month.class`).

The final constructor allows you to specify descriptions for the domain and range of the data:

> `public BasicTimeSeries(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.

18.4.3 Attributes

Each instance of `BasicTimeSeries` has the following attributes:

Attribute:	**Description:**
`Name`	The name of the series (inherited from `Series`).
`DomainDescription`	A description of the time period domain (for example, 'Quarter'). The default is 'Time'.
`RangeDescription`	A description of the value range (for example, 'Price'). The default is 'Value'.

18.4.4 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 TimeSeriesDataPair getDataPair(int item)`
> Returns a data item. The `item` argument is a zero-based index.

To retrieve a particular value from a series by time period:

> `public TimeSeriesDataPair getDataPair(TimePeriod period)`
> Returns the data item (if any) for the specified time period.

To add a value to a time series:

> `public void add(TimePeriod 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.

18.4.5 Notes

The class name was formerly `TimeSeries`, but this has been changed to avoid confusion with the subclass in the `com.jrefinery.finance` package.

See Also

`TimePeriod`, `TimeSeriesCollection`.

18.5 CategoryDataset

18.5.1 Overview

An interface (extending `SeriesDataset`) that defines the structure of a *category dataset*. The dataset consists of a table of series and categories. A value is associated with each combination of series and category (`null` values are permitted).

18.5.2 Methods

To obtain the number of categories:

> `public int getCategoryCount();`
> Returns the number of categories in the dataset.

To get a list of the categories in the dataset:

> `public List getCategories();`
> Returns a list of the categories in the dataset.

To get the value for a series/category combination:

> `public Number getValue(int series, Object category);`
> Returns the value associated with a particular series and category. The value may be `null`.

18.5.3 Notes

You can use any `Object` instance to represent a category. Using `String` is convenient, as the `toString()` method is used whenever a label is required for a category.

This interface is intended for reading data, not updating it.

See Also

`DefaultCategoryDataset`, `SeriesDataset`.

18.6 CombinationDataset

18.6.1 Overview

An interface for combining datasets. Written by Bill Kelemen.

18.6.2 Notes

This interface is used to create combined charts with the JFreeChart class library.

See Also

`CombinedDataset`.

18.7 CombinedDataset

18.7.1 Overview

An implementation of the `CombinationDataset` interface. Written by Bill Kelemen.

18.7.2 Notes

This class is used to create combined charts with the JFreeChart class library.

See Also

`CombinationDataset`.

18.8 Dataset

18.8.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by `PieDataset`, `CategoryDataset` and `SeriesDataset`.

18.8.2 Methods

A couple of methods relate to the use of datasets for drawing charts (see JFreeChart):

```
public int getLegendItemCount();
```
Returns the number of items to display in the legend.

```
public String[] getLegendItemLabels();
```
Returns an array of strings to use as labels in the legend.

Two further methods are used for registering change listeners with the dataset:

```
public void addChangeListener(DatasetChangeListener listener);
```
Registers a change listener with the dataset.

```
public void removeChangeListener(DatasetChangeListener listener);
```
Deregisters a change listener.

18.8.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as `PieDataset`, `CategoryDataset` or `XYDataset`.

See Also

`PieDataset`, `SeriesDataset`.

18.9 DatasetChangeEvent

18.9.1 Overview

An event that is used to provide information about changes to datasets.

See Also

`DatasetChangeListener`.

18.10 DatasetChangeListener

18.10.1 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 needs to implement this interface and register itself with the dataset.

18.10.2 Methods

The interface defines a single method:

```
public void datasetChanged(DatasetChangeEvent event);
```
Receives notification of a change to a dataset.

See Also

`DatasetChangeEvent`.

18.11 Datasets

18.11.1 Overview

A collection of utility methods for working with datasets.

18.11.2 Methods

To get the minimum and maximum domain values in a dataset:

```
public static Number getMinimumDomainValue(Dataset data);
```
Returns the minimum domain value for the dataset.

```
public static Number getMaximumDomainValue(Dataset data);
```
Returns the maximum domain value for the dataset.

To get the minimum and maximum range values in a dataset:

`public static Number getMinimumRangeValue(Dataset data);`
Returns the minimum range value for the dataset.

`public static Number getMaximumRangeValue(Dataset data);`
Returns the maximum range value for the dataset.

To create a `PieDataset` from a `CategoryDataset`:

`public static PieDataset createPieDataset(CategoryDataset data, Object category);`
Returns a pie dataset by taking all the values in the category dataset for the specified category.

`public static PieDataset createPieDataset(CategoryDataset data, int series) ;`
Returns a pie dataset by taking all the values in the category dataset for the specified series.

See Also

`DomainInfo`, `RangeInfo`.

18.12 Day

18.12.1 Overview

A subclass of `TimePeriod` that represents one day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.12.2 Constructor

To construct a `Day` instance:

`public Day(int day, int month, int year);`
Creates a new `Day` instance. The `year` argument should be in the range 1900 to 9999.

To create a `Day` instance based on a `SerialDate`:

`public Day(SerialDate day);`
Creates a new `Day` instance.

To create a `Day` instance based on a `Date`:

`public Day(Date time);`
Creates a new `Day` instance.

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.

18.12.3 Methods

To access the day:

> `public SerialDate getDay();`
> Returns the day as a `SerialDate`.

There is no method to *set* the day, because this class is immutable.

Given a `Day` object, you can create an instance representing the previous day or the next day:

> `public TimePeriod previous();`
> Returns the previous day, or null if the lower limit of the range is reached.
>
> `public TimePeriod 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.
>
> `public static Day parseDay(String s) throws TimePeriodFormatException;`
> Parses the string and, if possible, returns a `Day` object.

18.12.4 Notes

In the current implementation, the day can be in the range 1-Jan-1900 to 31-Dec-9999.

The `Day` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `SerialDate`.

18.13 DefaultCategoryDataset

18.13.1 Overview

A default implementation of the `CategoryDataset` interface that uses an array to store data.

18.13.2 Constructors

There are several constructors for this class.

> `public DefaultCategoryDataset(Number[][] data);`
> Constructs a dataset from an array. Default series names are generated in the form `Series 1`, `Series 2`, ... , `Series m`. Default categories are generated as `String` objects in the form `Category 1`, `Category 2`, ..., `Category n`.

`public DefaultCategoryDataset(String[] seriesNames, Object[] categories, Number[][] data);`
Constructs a dataset from an array. Also specified are the series names and the categories.

Note that this class is simply a wrapper around the data array supplied to the constructor. If you modify the contents of the array directly (using your original reference to it) then you by-pass the event notification mechanism used by this class to inform registered listeners of changes to the data. You shouldn't do this unless you are sure that is what you want to do.

18.13.3 Methods

To change the series names:

`public void setSeriesNames(String[] seriesNames);`
Changes all the series names for the dataset.

To change the categories:

`public void setCategories(Object[] categories);`
Changes all the categories for the dataset.

18.13.4 Notes

You can use any object to represent a category. The category label will be the value returned by the `toString()` method. Most of the time, the `String` class is sufficient for representing categories.

See Also
`CategoryDataset`

18.14 DefaultPieDataset

18.14.1 Overview

A convenient implementation of the `PieDataset` interface.

18.14.2 Constructors

The default constructor creates an empty pie dataset:

`public DefaultPieDataset();`
Creates a new dataset, initially empty.

`public DefaultPieDataset(Collection values);`
Creates a new dataset containing the values supplied. Section names are automatically generated.

18.14.3 Methods

To get the value for a particular category:

> `public Number getValue(Object category);`
> Returns the number associated with a category. This method can return `null`.

To set the value for a particular category:

> `public void setValue(Object category, Number value);`
> Sets the number associated with a category.

18.14.4 Notes

The dataset can contain `null` values.

See Also

`PieDataset`.

18.15 DefaultXYDataset

A quick and dirty implementation of the `XYDataset` interface. This class is in the process of being replaced by `XYSeriesCollection`.

See Also

`XYDataset`

18.16 DomainInfo

18.16.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's domain.

18.16.2 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.

18.16.3 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`.

18.17 HighLowDataset

An extension of the `XYDataset` interface, that supplies data in the form of "high/low, open/close" items.

`public Number getHighValue(int series, int item);`
Returns the high value for an item within a series.

`public Number getLowValue(int series, int item);`
Returns the low value for an item within a series.

`public Number getOpenValue(int series, int item);`
Returns the open value for an item within a series.

`public Number getCloseValue(int series, int item);`
Returns the close value for an item within a series.

This interface is used in the JFreeChart library.

18.18 Hour

18.18.1 Overview

A subclass of `TimePeriod` that represents one hour in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.18.2 Constructor

To construct an `Hour` instance:

`public Hour(int hour, Day day);`
Creates a new `Hour` instance. The `hour` argument should be in the range 0 to 23.

To construct an `Hour` instance based on a `java.util.Date`:

`public Hour(Date time);`
Creates a new `Hour` instance.

A default constructor is provided:

> `public Hour();`
> Creates a new `Hour` instance based on the current system time.

18.18.3 Methods

To access the hour:

> `public int getHour();`
> Returns the hour (in the range 0 to 23).

To access the day:

> `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 or the next hour:

> `public TimePeriod previous();`
> Returns the previous hour, or null if the lower limit of the range is reached.
>
> `public TimePeriod next();`
> Returns the next hour, or null if the upper limit of the range is reached.

18.18.4 Notes

The `Hour` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Day`.

18.19 IntervalXYDataset

An extension of the `XYDataset` interface. Additional methods are provided to define an interval around the X and Y values:

> `public Number getStartXValue(int series, int item);`
> Returns the starting x-value for an item within a series.
>
> `public Number getEndXValue(int series, int item);`
> Returns the ending x-value for an item within a series.
>
> `public Number getStartYValue(int series, int item);`
> Returns the starting y-value for an item within a series.
>
> `public Number getEndYValue(int series, int item);`
> Returns the ending y-value for an item within a series.

18.20 IntervalXYZDataset

A natural extension of the `IntervalXYDataset` interface.

18.21 Millisecond

18.21.1 Overview

A subclass of `TimePeriod` that represents one millisecond within a particular second. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.21.2 Constructors

To construct a `Millisecond`

18.21.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:

`TimePeriod`, `BasicTimeSeries`, `Second`.

18.22 Minute

18.22.1 Overview

A subclass of `TimePeriod` that represents one minute in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.22.2 Constructors

To construct a `Minute` instance:

> `public Minute(int minute, Hour hour);`
> Creates a new `Minute` instance. The `minute` argument should be in the range 0 to 59.

To construct a `Minute` instance based on a `java.util.Date`:

> `public Minute(Date time);`
> Creates a new `Minute` instance.

A default constructor is provided:

> `public Minute();`
> Creates a new `Minute` instance, based on the current system time.

18.22.3 Methods

To access the minute:

> `public int getMinute();`
> Returns the minute (in the range 0 to 59).

To access the hour:

> `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 or the next minute:

`public TimePeriod previous();`
Returns the previous minute, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next minute, or null if the upper limit of the range is reached.

18.22.4 Notes

The `Minute` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Day`.

18.23 Month

18.23.1 Overview

A subclass of `TimePeriod` that represents one month in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.23.2 Constructors

To construct a `Month` instance:

`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.

To construct a `Month` instance based on 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.

18.23.3 Methods

To access the month:

`public int getMonth();`
Returns the month (in the range 1 to 12).

To access the year:

> `public Year getYear();`
> Returns the year.

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 or the next month:

> `public TimePeriod previous();`
> Returns the previous month, or null if the lower limit of the range is reached.
>
> `public TimePeriod 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.

18.23.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Month` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

18.24 PieDataset

18.24.1 Overview

The interface for a dataset that associates values with categories.

18.24.2 Methods

Three methods are defined in the interface:

> `public int getCategoryCount();`
> Returns the number of categories in the dataset.
>
> `public Set getCategories();`
> Returns the set of categories.
>
> `public Number getValue(Object category);`
> Returns the value associated with a particular category.

18.24.3 Notes

The name of the interface is derived from a common usage for this type of dataset—the creation of pie charts.

There are some convenient methods for creating a `PieDataset` object by slicing a `CategoryDataset`. Refer to the `Datasets` class for more details.

See Also

`DefaultPieDataset`.

18.25 Quarter

18.25.1 Overview

A subclass of `TimePeriod` that represents one quarter in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.25.2 Constructor

To construct a `Quarter` instance:

```
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.

To construct a `Quarter` instance based on 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.

18.25.3 Methods

To access the quarter:

```
public int getQuarter();
```
Returns the quarter (in the range 1 to 4).

To access the year:

```
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 quarter or the next quarter:

> `public TimePeriod previous();`
> Returns the previous quarter, or null if the lower limit of the range is reached.
>
> `public TimePeriod 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.

18.25.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Quarter` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

18.26 RangeInfo

18.26.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's range.

18.26.2 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.

18.26.3 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`.

18.27 Second

18.27.1 Overview

A subclass of `TimePeriod` that represents one second in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.27.2 Constructors

To construct a `Second` instance:

```
public Second(int second, Minute minute);
```
Creates a new `Second` instance. The `second` argument should be in the range 0 to 59.

To construct a `Second` instance based on 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.

18.27.3 Methods

To access the second:

```
public int getSecond();
```
Returns the second (in the range 0 to 59).

To access the Minute:

```
public Minute getMinute();
```
Returns the minute.

There is no method to *set* the second or the day, because this class is immutable.

Given a `Second` object, you can create an instance representing the previous second or the next second:

> `public TimePeriod 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.

18.27.4 Notes

The `Second` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:

`TimePeriod`, `BasicTimeSeries`, `Day`.

18.28 SeriesChangeListener

The interface through which series change notifications are posted.

18.29 SeriesDataset

18.29.1 Overview

A base interface that defines a dataset containing zero, one or many data series.

18.29.2 Methods

The methods in the interface are:

> `public int getSeriesCount();`
> Returns the number of series in the dataset.
>
> `public String getSeriesName(int series);`
> Returns the name of the series with the specified index (zero based).

18.29.3 Notes

This interface is extended by `CategoryDataset` and `XYDataset`.

See Also:

`CategoryDataset`, `XYDataset`.

18.30 SeriesException

An exception generated 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`.

18.31 Statistics

18.31.1 Overview

Provides some static utility methods for calculating statistics.

18.31.2 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.

18.31.3 Notes

This class was written by Matthew Wright.

18.32 SubseriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

18.33 TimePeriod

18.33.1 Overview

An abstract class that represents a period of time. A number of concrete subclasses have been implemented: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

The time periods represented by this class are not (in general) fixed to a particular time zone. No matter where you are in the world, if you create a new `Day` object to represent `1-Apr-2002`, that is the day it represents.

Of course, against a real time line, `1-Apr-2002` in (say) New Zealand is not the same as `1-Apr-2002` in France. But sometimes you want to treat them as if they were the same. 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.

In a sense, the time period classes are designed to be imprecise.

Occasionally you may want to convert a `TimePeriod` 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 `TimePeriod` instance to a particular time zone. The various `getStart(...)` and `getEnd(...)` methods provide this facility, using the default timezone, a user supplied timezone, or a `Calendar` with the timezone preset.

18.33.2 Methods

Given a `TimePeriod` instance, you can create another instance representing the previous time period, or the next time period:

> `public abstract TimePeriod previous();`
> Returns the previous time period, or `null` if the current time period is the first in the supported range.
>
> `public abstract TimePeriod 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 getStart();`
> Returns the first millisecond of the time period, evaluated using the default timezone.
>
> `public long getStart(TimeZone zone);`
> Returns the first millisecond of the time period, evaluated using a particular timezone.
>
> `public abstract long getStart(Calendar calendar);`
> Returns the first millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).
>
> `public long getMiddle();`
> Returns the middle millisecond of the time period, evaluated using the default timezone.
>
> `public long getMiddle(TimeZone zone);`
> Returns the middle millisecond of the time period, evaluated using a particular timezone.

`public long getMiddle(Calendar calendar);`
Returns the middle millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

`public long getEnd();`
The last millisecond of the time period, evaluated using the default timezone.

`public long getEnd(TimeZone zone);`
Returns the last millisecond of the time period, evaluated using a particular timezone.

`public abstract long getEnd(Calendar calendar);` Returns the last millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

18.33.3 Notes

This class and its subclasses can be used with the `BasicTimeSeries` class.

All `TimePeriod` subclasses are required to be immutable.

Known subclasses include: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

See Also:
`BasicTimeSeries`.

18.34 TimePeriodFormatException

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

See Also
`TimePeriod`

18.35 TimeSeriesCollection

18.35.1 Overview

A collection of `TimeSeries` objects. The collection may contain zero, one or many time series.

`TimeSeriesCollection` extends `AbstractSeriesDataset` to provide some of the basic series information.

The collection implements the `IntervalXYDataset` interface (and, therefore, the `XYDataset` interface) and can be used as a convenient dataset for the JFreeChart library.

18.35.2 Constructors

You can construct a `TimeSeriesCollection` in several different ways:

> `public TimeSeriesCollection();`
> Creates a new time series collection, initially empty.
>
> `public TimeSeriesCollection(BasicTimeSeries 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. There are not yet any methods for removing a series from a collection (possibly to be implemented in the future).

18.35.3 Methods

To find out how many time series objects are in the collection:

> `public int getSeriesCount();`
> Returns the number of time series objects in the collection.

To get a reference to a particular series:

> `public BasicTimeSeries 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(BasicTimeSeries 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 part of the `XYDataset` interface.

18.35.4 Notes

This class implements the `XYDataset` and `IntervalXYDataset` interfaces.

See Also:
`AbstractSeriesDataset`, `BasicTimeSeries`, `XYDataset` and `IntervalXYDataset`.

18.36 TimeSeriesDataPair

Associates a numerical value with a time period. This class is used by the `TimeSeries` class.

There are a number of important features. First, the class implements the `Comparable` interface, allowing data items to be sorted into time order using standard Java API calls. Second, the instances of this class can be easily cloned. Third, 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.

See Also

`TimeSeries`

18.37 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

18.38 Values

An interface for accessing a set of values. This hasn't been used for anything yet...but the idea was to create a simple data structure that could be passed to a variety of statistical methods (for example, a method that calculates frequency distributions, returning an appropriate dataset for constructing a histogram). More work to be done...

18.39 Week

18.39.1 Overview

A subclass of `TimePeriod` that represents one week in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

18.39.2 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.

18.39.3 Methods

To access the week:

> `public int getWeek();`
> Returns the week (in the range 1 to 52).

To access the year:

> `public Year getYear();`
> Returns the year.

There is no method to *set* the week or the year, because this class is immutable.

Given a `Week` object, you can create an instance representing the previous week or the next week:

> `public TimePeriod previous();`
> Returns the previous week, or null if the lower limit of the range is reached.

> `public TimePeriod next();`
> Returns the next week, or null if the upper limit of the range is reached.

To convert a `Week` object to a `String` object:

> `public String toString();`
> Returns a string representing the week.

18.39.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Week` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

18.40 XYDatapair

Associates a numerical value with another numerical value. This class is analagous to the `TimeSeriesDataPair` class.

18.41 XYDataset

18.41.1 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. Each series can have *(x, y)* values that are completely independent of the other series in the dataset.

18.41.2 Methods

The methods in the interface are:

`public int getItemCount(int series);`
Returns the number of data items in a series.

`public Number getXValue(int series, int item);`
Returns an x-value for a series.

`public Number getYValue(int series, int item);`
Returns a y-value for a series (possibly `null`).

18.41.3 Notes

JFreeChart uses this interface to obtain data for drawing charts.

See Also:
`SeriesDataset`, `DefaultXYDataset`, `IntervalXYDataset`.

18.42 XYSeries

18.42.1 Overview

A series of (x, y) data items. Each item is represented by an instance of `XYDataItem` and stored in a list.

18.42.2 Constructors

To construct a series:

`public XYSeries(String name);`
Creates a new series (initially empty) with the specified name.

18.42.3 Methods

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 are not allowed.

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 find out how many items are contained in a series:

`public int getItemCount();`
Returns the number of items in the series.

18.42.4 Notes

This class extends `Series`, so you can register change listeners with the series.

You can create a collection of series using the `XYSeriesCollection` class. Since `XYSeriesCollection` implements the `XYDataset` interface, this is a convenient structure for supplying data to JFreeChart.

See Also:
`XYSeriesCollection`.

18.43 XYSeriesCollection

18.43.1 Overview

A collection of `XYSeries` objects. This class implements the `XYDataset` interface, so can be used very conveniently with JFreeChart.

18.43.2 Constructors

To construct a series collection:

`public XYSeriesCollection();`
Creates a new empty series collection.

18.43.3 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.

18.43.4 Notes

This class implements the `XYDataset` interface, so it is a convenient class for use with JFreeChart.

See Also:
`XYSeries`.

18.44 XYZDataset

A natural extension of the `XYDataset` interface.

18.45 Year

18.45.1 Overview

A subclass of `TimePeriod` that represents one year. This class is designed to be used with the `TimeSeries` class, but is (hopefully) general enough to be used in other situations.

18.45.2 Constructors

To construct a `Year` instance:

> `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.

18.45.3 Methods

To access the year:

> `public int getYear();`
> Returns the year.

There is no method to *set* the year, because this class is immutable.

Given a `Year` object, you can create an instance representing the previous year or the next year:

`public TimePeriod previous();`
Returns the previous year, or null if the lower limit of the range is reached.

`public TimePeriod 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, or vice versa:

`public String toString();`
Returns a string representing the year.

`public static Year parseYear(String s) throws TimePeriodFormatException;`
Parses the string and, if possible, returns a `Year` object.

18.45.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Year` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `TimeSeries`.

A The GNU Lesser General Public Licence

A.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public Licence (LGPL). The full text of this licence is reproduced in this appendix. You should read and understand this licence before using JFreeChart in your own projects.

If you are not familiar with the idea of *free software* and/or *open source software*, you can find out more at the following web-sites:

Organisation:	**Description:**
The Free Software Foundation	`http://www.fsf.org`
The Open Source Initiative	`http://www.opensource.org`

Please send e-mail to `david.gilbert@object-refinery.com` if you have any questions about the licensing of JFreeChart.

A.2 The Licence

The following licence has been used for the distribution of the JFreeChart class library:

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

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

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 o□er, 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 o□ering access to copy from a designated place, o□er 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