Software Reengineering Project

Mitchel Pyl & Randy Paredis

1 Introduction

This document is meant as additional information on the reengineering and refactoring of the `JFreeChart` project, which was the assignment of the Software Reengineering course of 2019, at the University of Antwerp.

During this paper and the process of reengineering the `JFreeChart` project, we relied heavily on [10], making sure we could complete our assignment as successful as possible.

`JFreeChart` [8] is a Java library that can be used to add/show professional-looking graphs and charts in your Java applications. This inheritly implies that is it useful in a lot of different contexts and scenarios that require this kind of feature.

The ability for such a library for being flexible and expandable with a vast amount of new features would therefore be an incredible advantage for this.

1.1 Problem at Hand

At this point in time, `JFreeChart` has a wide range of possible graphs, charts and plots it can generate for any kind of data you'd like. However, there is some functionality missing that we'd like to have. Namely, we'd like to be able to have a different shape or symbol for each datapoint. In order for us to introduce this feature, we'll first have to figure out the current way rendering of datapoints is handeled and afterwards we'll refactor the code so we can easily add this feature.

Additionally, when we take a closer look at the code in general, there are some symptoms indicating it should be refactored[1].

[1] *Missing tests*, *Too much time for simple changes...*; 1.1 from [10])

2 Project Management

2.1 Setting Direction

The most important aspect of managing a reengineering project is to find a strategy in which the reengineering will be the most useful and succesful (Chapter 2 from [10]). This is why we first discussed a strategy to use in the actual reengineering, before jumping into the code like headless chickens.

Using some tools, we were able to *Agree on Maxims* (2.1 from [10]) and more specifically the *Most Valuable First* (2.4 from [10]). With these strategies in mind, we can give all refactoring targets a weight, so we can easily list the most important ones. As described in 2.4 from [10], such a weight technically has nothing to do with cyclic complexities, but with what's valuable to the customer. In our case, these luckily (or coincidentally) line up to a certain point.

Learning the most important rule in software reengineering, *If It Ain't Broke, Don't Fix It.* (2.6 from [10])[2], we know we'd best not touch any code that is "working" correctly and has nothing to do with any of the valuable targets. For instance, within the scope of the assignment, it is not useful to take a look at refactoring the `ImageMapUtils`.

While on the topic, although all strategies have their merit and are important in some way, we believe some of them are more important and/or practical to follow. *Keep It Simple* (2.7 from [10]) is one of them, which we will keep in mind during the refactoring process.

2.2 PERT-Chart

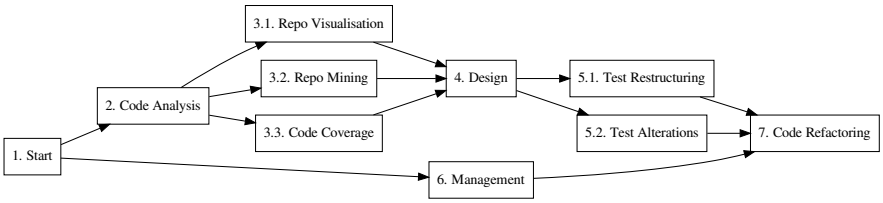


Figure 1: Simplified PERT-Chart for the Refactoring of `JFreeChart`

[2]Note: this is a rule, not a lifeline, nor an excuse. (as per [10])

In order for us to cleanly work on the reengineering of `JFreeChart`, we decided to make a PERT-chart [11], as you can see in *Fig. 1*. This is a simplified model, without annotations of any critical tasks, paths, or the latest end dates for each task.

As you can see, *Management* is a task we will do throughout the entire process of refactoring the project. Tasks *3.1. Repo Visualisation*, *3.2 Repo Mining* and *3.3 Code Coverage* can be found later in this document, respectively in *sections 3.1, 3.2* and *3.3*.

2.3 First Contact

Seeing as we don't have any experience with the project, nor with its uses, we tried to get a good, general overview. When trying to *Read all the Code in One Hour* (3.2 from [10]) and to *Skim the Documentation* (3.3 from [10]), we found that is was quite the difficult task and although it certainly helped in getting a good understanding of how the system worked, it was far from ideal.

3 Project Analysis and Tool Usage

In order to solidly identify the issues with `JFreeChart` and find possible refactoring targets, we made use of a few helpful tools that allowed for clear identification of possible problem areas. This way we can clearly *Study the Exceptional Entities* (4.3 from [10]). Even though we know that what we find may be tedious or ambiguous to interpret, we will still make our conclusions based on what we know and expect.

3.1 Repository Visualization with Gource

The first tool we made use of was Gource [6]. It is a clean and fancy piece of software that can turn the history of a git repository into a visual representation. This is useful for a few reasons. First, it allows us to see clearly who the main contributers are. There was no surprise that this was *David Gilbert*.

A second thing we could deduce from this simulation is that the code was not made using the Test Driven Development methodology. We can clearly see that there are first adaptations to the codebase, before changing the tests.

Thirdly, we can identify the possible points in time when a refactoring stage happened in this project. These are moments when a lot of files are

added, removed, or modified; which is highlighted in Gource. Granted, it is possible that some of these changes are due to merging multiple branches together.

We've identified that possible refactorings happened in November 2008 (increase in functionality), March 2013 (update to almost all files), December 2014 (update to almost all files and removal of a lot of files), July 2017 (file tree restructuring) and July 2018 (general changes).

Finally, we can use the resulting visualization to *Learn from the Past* (5.5 from [10]). We can see which classes were changed a lot and which ones remained untouched for the main bulk of the development.

Classes that changed a lot most likely indicate that they are coupled to other classes in the system, marking these classes as important in understanding the general feel of the system.

Classes that remained mainly untouched could indicate abandoned code, but let's assume another possibility. Let's say that these classes indicate features or functionality that are complete. Usually these features cannot give a good enough representation of what's important in the system and whatnot. In either case of untouched code, we can remove our focus from these classes.

3.2 Repository Mining with CodeScene

Another tool we made use of was CodeScene [5], the powerful visualization tool using *Predictive Analytics* to find hidden risks and social patterns in your code.

CodeScene allowed us to get a general feel of the current state of `JFreeChart`. It gave us a clear representation of possible refactoring targets (see *attachment 7*) and hotspots (see *attachment 4*) within the code[3].

When we take a deeper look into the code (or at least the graphical representation thereof), we can identify that we most probably will need to take a look at the `org.jfree.chart.renderer` package (see *attachment 5*) and the `org.jfree.chart.plot` package (see *attachment 6*), as far as the hotspots are concerned.

On the topic of refactoring targets, it is clear that the `org.jfree.chart.plot` package (see *attachment 8*) really inquires our attention. More specifically the `XYPlot` (*attachment 10*), `CategoryPlot` (*attachment 11*), `PiePlot` (*attachment 12*), `AbstractXYItemRenderer` (*attachment 13*) and `AbstractCategoryItemRenderer` (*attachment 14*) classes. In the attach-

[3]Please refer to the attachments at the end of this document.

ments, the most complex functions are listed (sorted from high to low complexity). These top functions[4] are most likely to be refactoring targets.

3.3 Code Coverage with Cobertura

Chapter 6.3 of [10] tells us to *Use a Testing Framework*. Not only is this a good idea in refactoring, but in all software projects in general. JUnit was already available in `JFreeChart`, so there is no need to change of alter this part of the project. Linked with JUnit was Cobertura [4], a maven plugin that allows us to check how much code was covered with the available tests.

The overview that is generated from this plugin gives us enough information in order to determine which classes and functions were not covered in the project, also yielding possible missing tests. These missing tests can be seen as a symptom for code requiring refactoring (1.1 from [10]). But as discussed above, we will *Fix Problems, Not Symptoms* (2.5 from [10]) and more specifically, we will mainly focus on the tests that concern our main refactoring targets.

In general, we can deduce that the code coverage of `JFreeChart` at this point in time is way below comfortable for us.

We also noticed that there is currently no mutation testing being done on this project. Even though we do realize this would give way too much situations and possibilities to cover, we currently have no idea of how good the tests currently are.

3.4 IntelliJ/Eclipse

Because of the jumbled mess `JFreeChart` is, we decided to look into some tools that might help with the refactoring process itself. The first one that came to mind (and the one that we used to the biggest extend) were the IntelliJ refactoring functionalities (extracting methods/classes, pulling functions up/down...).

Due to our usage with IntelliJ, we also installed the Code Smells Detector plugin [2]. Unfortunately, this tool appears to be sort of buggy when actually trying to perform some refactoring from the builtin functionality. This is why we combined this plugin with the CodeMetrics plugin [3], so we could obtain a valid annotation on the complexity of some functions that were highlighted by the Code Smells Detector. The functions we found here (mostly) lined up with the one we obtained from CodeScene, giving us additional confirmation.

[4]The ones with a red cyclomatic complexity.

The Eclipse counterpart of this plugin would be JDeodorant [1], but sadly it was quite confusing to use and the results it produced (extracting a few methods to a superclass, moving a method to the class that is used the most in that method...) were, in our opinion, not helpful whatsoever. This is why we decided to identify most of the methods that needed extracting ourselves first and checking the result again afterwards, to see if we have made a difference.

3.5 Code Clones with iClones

While on the topic of code smells, we also used iClones so we could detect duplicate code and other aspects outside of the IDEs we're using. Seeing iClones has builtin functionality to compare different versions of a project, it allows us to easily compare the code at the start of our refactoring process and at the end[5].

Since the results we obtain from using this software are most useful in comparison over time, we will defer our conclusions to *section 5*.

4 Refactoring

4.1 Design Recovery

In order for us to describe our proposed new design of `JFreeChart`, we must first tell you a little bit more about the original design.

`JFreeChart` consisted of 658 classes, the multitude of which have nothing to do with our problem domain. When we only focus on the classes that are important (as per 2.4 and 2.6 from [10]), we can create a UML diagram. But because the way `JFreeChart` is currently designed, the UML class diagram of the subsection we will describe is incredibly wide and messy to look at. This is why we adapted the UML specification to get a clearer picture of what is actually happening (see *Fig. 3*).

[5] Although we could easily apply this somewhere in the middle of our refactoring process, we've decided that this information is not entirely useful within the context of what we're trying to obtain.

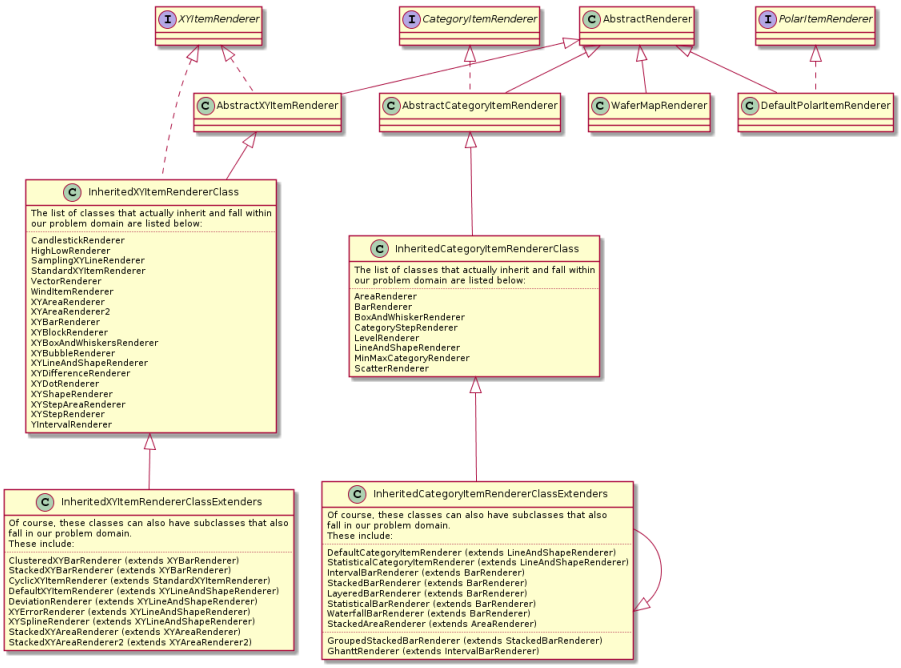


Figure 2: A simplified representation of the UML class diagram at the beginning of the refactoring process. Image created with [9]. A bigger version can be found in *attachment 2*.

The `InheritedXYItemRendererClass` in our UML class diagram represents any subclass of the `AbstractXYItemRenderer`. The same was done for the `InheritedCategoryItemRenderer` and the `AbstractCategoryItemRenderer`. In the `Extenders`, the few classes that extend from the `InheritedRenderers` are listed in the same way[6].

This expansion of the UML specification allowed us to generally show the structure of our problem domain. Yet, trying to identify this domain, or more specifically, the functionality that required refactoring was quite tricky. This is mainly due to the high rate of exceptional entities and anomalies we've found while applying the *Study the Exceptional Entities* pattern (4.3 from [10]).

Some of these anomalies[7], we've listed below.

- Renderers that do not render;

[6]Though, for the `Category` case there is another layer in the structure, hence the self-referring arrow in the `InheritedCategoryItemRendererClassExtenders`.

[7]As we would describe them. In [10] they are not mentioned

- Implicit implementation of interfaces due to extension of a class that also implements the interface;
- A mixture of a multitude of design patterns that are not all fully implemented (Abstract Factory, Adapters, Bridges, Model View Controller, Chain of Responsibility, Observers...)
- Groups of renderers that have similar features are completely separated. (e.g. the `AreaRenderer` and the `XYItemRenderer` have no correlation whatsoever)
- ...

4.2 Design

Based upon the design we recovered, we can generate a new, better (and hopefully cleaner) design.

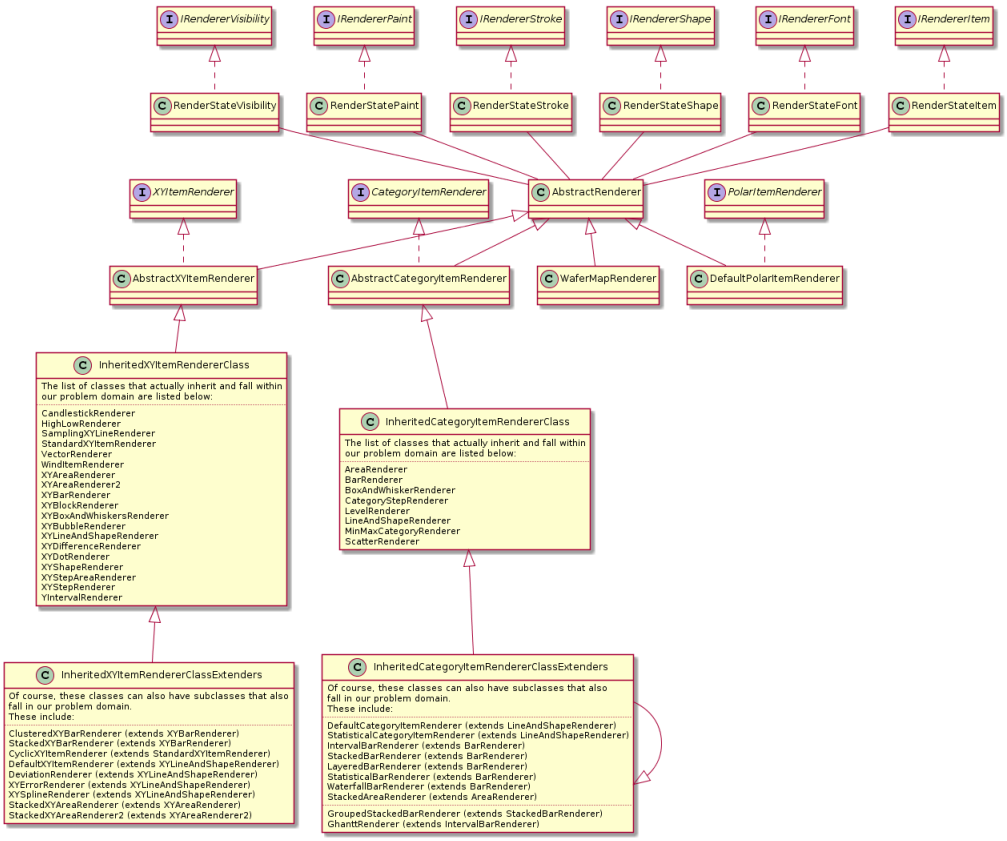


Figure 3: An UML diagram, showing how we see the design. Image created with [9]. A bigger version can be found in *attachment 3*.

First things first, we can see that all methods that extend from the `AbstractXYItemRenderer` implement the `XYItemRenderer` interface, that is also implemented by their base class. From the Liskov Substitution Principle and the basic rules of inheritance, we know that it is not required for derived classes to also implement the interface that was implemented by the base class. Even stronger, we know that all of the functions that are required by this interface are implemented in the derived classes, because they are available in the base class. This is why we severed the implementation link of all classes that inherit from the `AbstractXYItemRenderer`.

Unfortunately, without crowding too much of the class diagram, we didn't add all functions that are used in each class. If we would have

done that, it would become clear that the `AbstractRenderer` is, in fact, a god class. *Split Up God Class* (9.3 from [10]) tells us it is optimal to split up a class that holds too many functionalities, which is the case for the `AbstractRenderer`.

From the comments in the code of the `AbstractRenderer`, we found that it had thirteen main functionalities, split up in vague sections: `Series Visibility`, `Series Visibility in Legend`, `Item Label Visibility`, `Paint`, `Fill Paint`, `Outline Paint`, `Item Label Paint`, `Stroke`, `Outline Stroke`, `Shape`, `Item Label Font`, `Positive Item Label Position` and `Negative Item Label Position`. Upon inspection, we found that not all extending classes used all these functions, which was an additional sign for us to split them up this way. We identified six categories from the original thirteen sections: `Visibility`, `Paint`, `Stroke`, `Shape`, `Font` and `Item`. Each of these categories are identified with the `RenderState*`-classes and their respective interfaces.

4.3 Management

4.4 Refactoring

5 Preserved Behaviour

Of course, in order to be able to say that our refactoring process was effective, we must compare our results to the original source code. While tools like iClones [7] automatically come bundled with a version comparison, others, unfortunately, do not. This is why, for this phase of the project, we decided to jump back and forth to the original version of our project and bundle all of our findings below.

As we described in *section 3.3*, we've been using the tests from `JFreeChart`. As you can see, our project still passes on all of them, meaning that we (at least) provide the same functionality as when we started (6.1 and 7.6 from [10]).

Taking a look at the code coverage reports from Cobertura, we can clearly see that we have some subtle differences. These changes can be either positive (the percentages went up) or negative (they went down). See *attachments 8 and 1*

For the positive changes in these reports, we know they are a good thing. Generally, this implies the refactoring we've done caused the tests to cover more code, giving us the information that there is more code that has passed the tests.

The negative changes on the other hand can also be seen as favorable. The main reason for this is because we removed some lines of code from these files and placed them elsewhere, while the existing covered functionality remained.

References

[1] Eclipse jdeodorant. https://marketplace.eclipse.org/content/jdeodorant. Accessed: 2019-05-23.

[2] Jetbrains code smell detector. https://plugins.jetbrains.com/plugin/10778-code-smell-detector. Accessed: 2019-05-23.

[3] Jetbrains codemetrics. https://plugins.jetbrains.com/plugin/12159-codemetrics. Accessed: 2019-05-23.

[4] Cobertura homepage. https://cobertura.github.io/cobertura/. Accessed: 2019-04-24.

[5] CodeScene homepage. https://codescene.io/. Accessed: 2019-04-24.

[6] Gource homepage. https://gource.io/. Accessed: 2019-04-24.

[7] iClones homepage. http://www.softwareclones.org/iclones.php. Accessed: 2019-05-23.

[8] JFreeChart homepage. http://www.jfree.org/jfreechart/. Accessed: 2019-04-24.

[9] PlantUML homepage. http://plantuml.com. Accessed: 2019-05-26.

[10] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-oriented Reengineering Patterns*. Square Bracket Associates, 2009.

[11] U.S. Department of the Navy, Bureau of Naval Weapons, and Special Projects Office. PERT (Program Evaluation Research Task), July 1958.

Attachments

On the following pages, we've included a set of screenshots from CodeScene that are referred to in the previous sections. The attachment number is overlayed over the image.

Packages

All
org.jfree.chart
org.jfree.chart.annotations
org.jfree.chart.axis
org.jfree.chart.block
org.jfree.chart.date
org.jfree.chart.editor
org.jfree.chart.encoders
org.jfree.chart.entity
org.jfree.chart.event
org.jfree.chart.imagemap
org.jfree.chart.labels
org.jfree.chart.needle
org.jfree.chart.panel
org.jfree.chart.plot
org.jfree.chart.plot.dial
org.jfree.chart.renderer
org.jfree.chart.renderer.category

All Packages

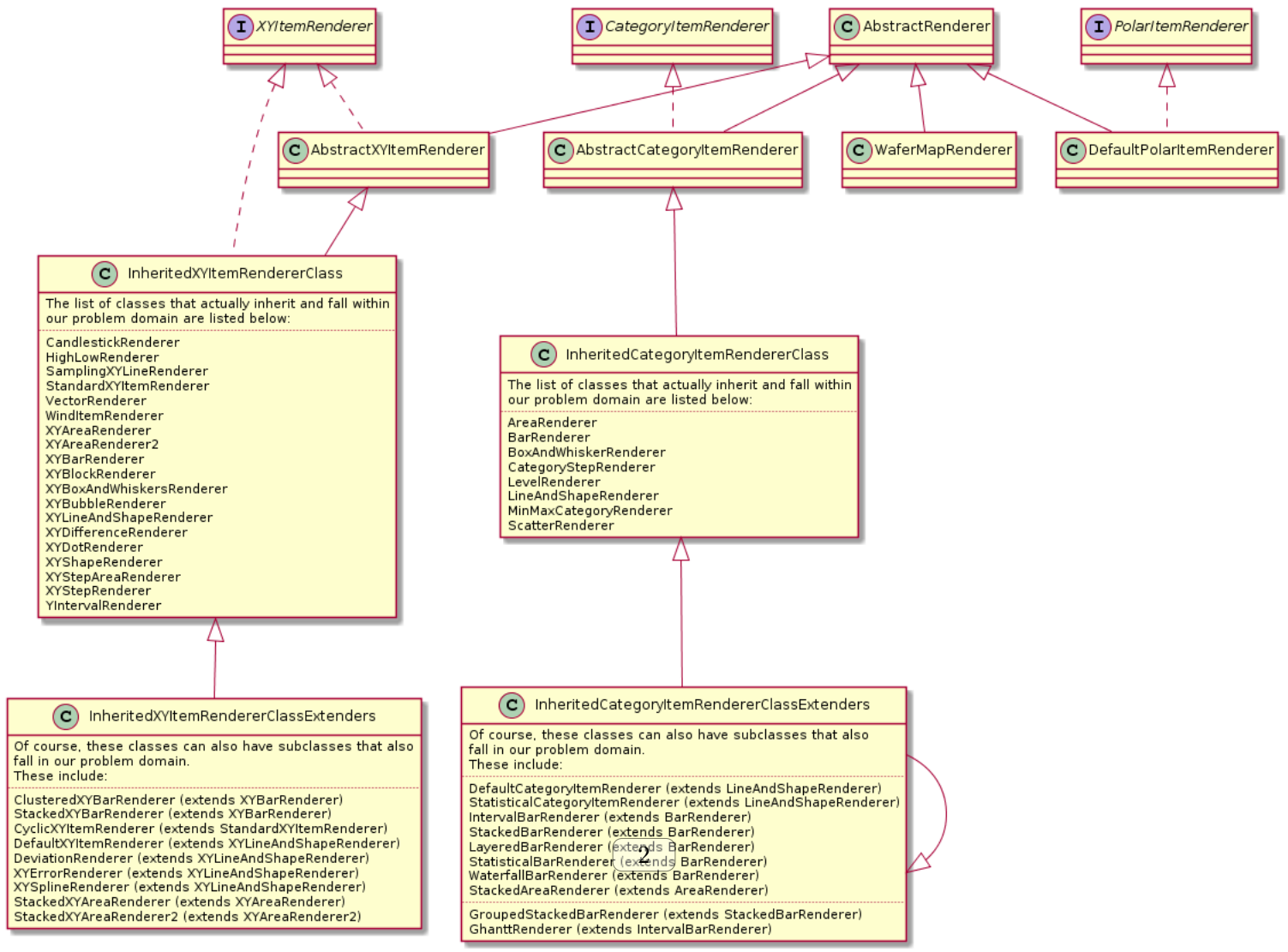
Classes

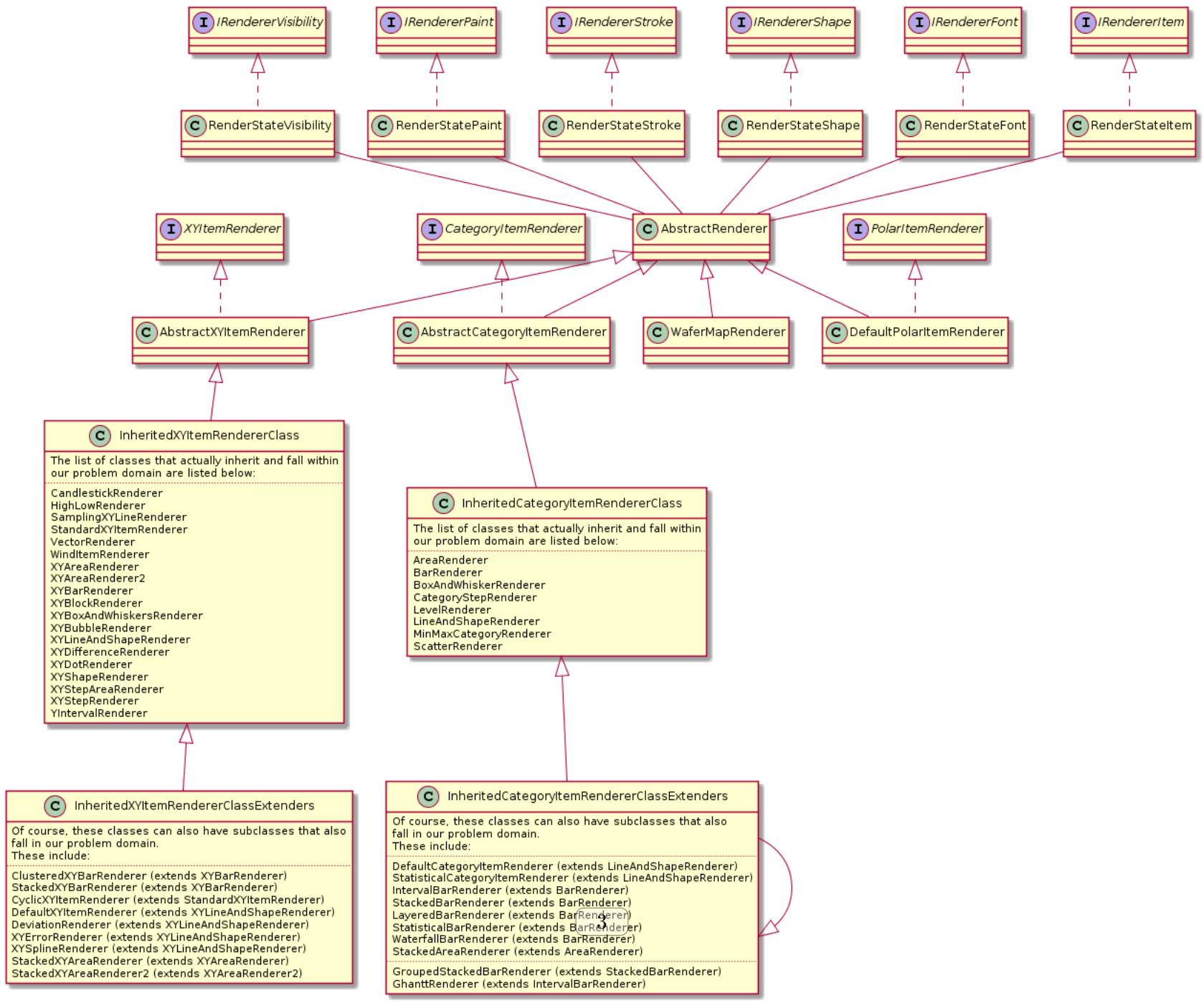
AbstractAnnotation *(77%)*
AbstractBlock *(88%)*
AbstractCategoryItemLabelGenerator *(87%)*
AbstractCategoryItemRenderer *(51%)*
AbstractDataset *(79%)*
AbstractDialLayer *(73%)*
AbstractIntervalXYDataset *(100%)*
AbstractObjectList *(89%)*
AbstractOverlay *(52%)*
AbstractPieItemLabelGenerator *(96%)*
AbstractPieLabelDistributor *(100%)*
AbstractRenderer *(82%)*
AbstractSeriesDataset *(100%)*
AbstractXYAnnotation *(41%)*
AbstractXYDataset *(100%)*
AbstractXYItemLabelGenerator *(80%)*
AbstractXYItemRenderer *(42%)*
AbstractXYZDataset *(16%)*
Align *(0%)*
Annotation *(N/A)*
AnnotationChangeEvent *(80%)*
AnnotationChangeListener *(N/A)*
ApplicationFrame *(0%)*
ArcDialFrame *(62%)*
AreaRenderer *(32%)*
AreaRendererEndType *(100%)*
Args *(80%)*
Arrangement *(N/A)*
ArrayUtils *(26%)*
ArrowNeedle *(30%)*
AttrStringUtils *(2%)*
AttributedStringUtils *(85%)*
Axis *(74%)*
AxisChangeEvent *(100%)*
AxisChangeListener *(N/A)*
AxisCollection *(90%)*
AxisEntity *(21%)*
AxisLabelLocation *(100%)*
AxisLocation *(27%)*
AxisSpace *(63%)*
AxisState *(59%)*
BarPainter *(N/A)*
BarRenderer *(65%)*

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	658	57% 29768/51388	46% 9864/21320	2.67
org.jfree.chart	25	50% 1711/3418	36% 432/1191	2.474
org.jfree.chart.annotations	20	59% 830/1395	52% 209/400	2.389
org.jfree.chart.axis	47	58% 3144/5392	43% 973/2252	3.025
org.jfree.chart.block	21	70% 873/1236	60% 314/516	2.81
org.jfree.chart.date	3	41% 96/230	24% 41/168	2.94
org.jfree.chart.editor	12	0% 0/839	0% 0/218	2.098
org.jfree.chart.encoders	6	0% 0/90	0% 0/6	1.212
org.jfree.chart.entity	14	38% 168/436	29% 46/154	2.071
org.jfree.chart.event	19	74% 47/63	N/A N/A	1
org.jfree.chart.imagemap	7	51% 42/81	64% 22/34	1.944
org.jfree.chart.labels	36	59% 580/976	45% 163/358	2.237
org.jfree.chart.needle	10	31% 124/392	34% 51/148	2.703
org.jfree.chart.panel	3	12% 34/267	3% 6/154	3.821
org.jfree.chart.plot	45	69% 6861/9924	56% 2287/4074	2.781
org.jfree.chart.plot.dial	18	59% 806/1357	61% 202/330	1.95
org.jfree.chart.renderer	15	63% 923/1449	50% 323/638	2.462
org.jfree.chart.renderer.category	27	55% 2035/3689	36% 617/1698	3.049
org.jfree.chart.renderer.xy	44	41% 2223/5353	28% 700/2425	3.072
org.jfree.chart.resources	1	0% 0/3	N/A N/A	1
org.jfree.chart.text	9	75% 489/647	53% 158/296	3
org.jfree.chart.title	10	58% 702/1199	38% 191/492	2.744
org.jfree.chart.ui	24	30% 223/721	33% 112/338	2.554
org.jfree.chart.urls	13	78% 256/325	64% 96/148	3.143
org.jfree.chart.util	32	48% 773/1608	40% 318/784	3.605
org.jfree.data	27	80% 884/1096	75% 399/530	2.811
org.jfree.data.category	7	81% 325/400	66% 152/230	2.882
org.jfree.data.function	5	80% 62/77	58% 14/24	1.92
org.jfree.data.gantt	6	66% 307/462	58% 98/168	2.022
org.jfree.data.general	26	72% 957/1312	67% 483/714	3.244
org.jfree.data.io	1	0% 0/57	0% 0/20	2.667
org.jfree.data.jdbc	3	0% 0/245	0% 0/131	4.452
org.jfree.data.json	1	0% 0/70	0% 0/20	4.5
org.jfree.data.json.impl	5	0% 0/185	0% 0/111	5.722
org.jfree.data.resources	6	16% 3/18	N/A N/A	1
org.jfree.data.statistics	18	75% 1033/1371	61% 394/638	2.54
org.jfree.data.time	27	64% 1604/2489	49% 481/978	2.373
org.jfree.data.time.ohlc	4	67% 105/155	66% 33/50	1.854
org.jfree.data.xml	9	0% 0/213	0% 0/56	2.087
org.jfree.data.xy	52	72% 1548/2148	65% 544/826	2.063

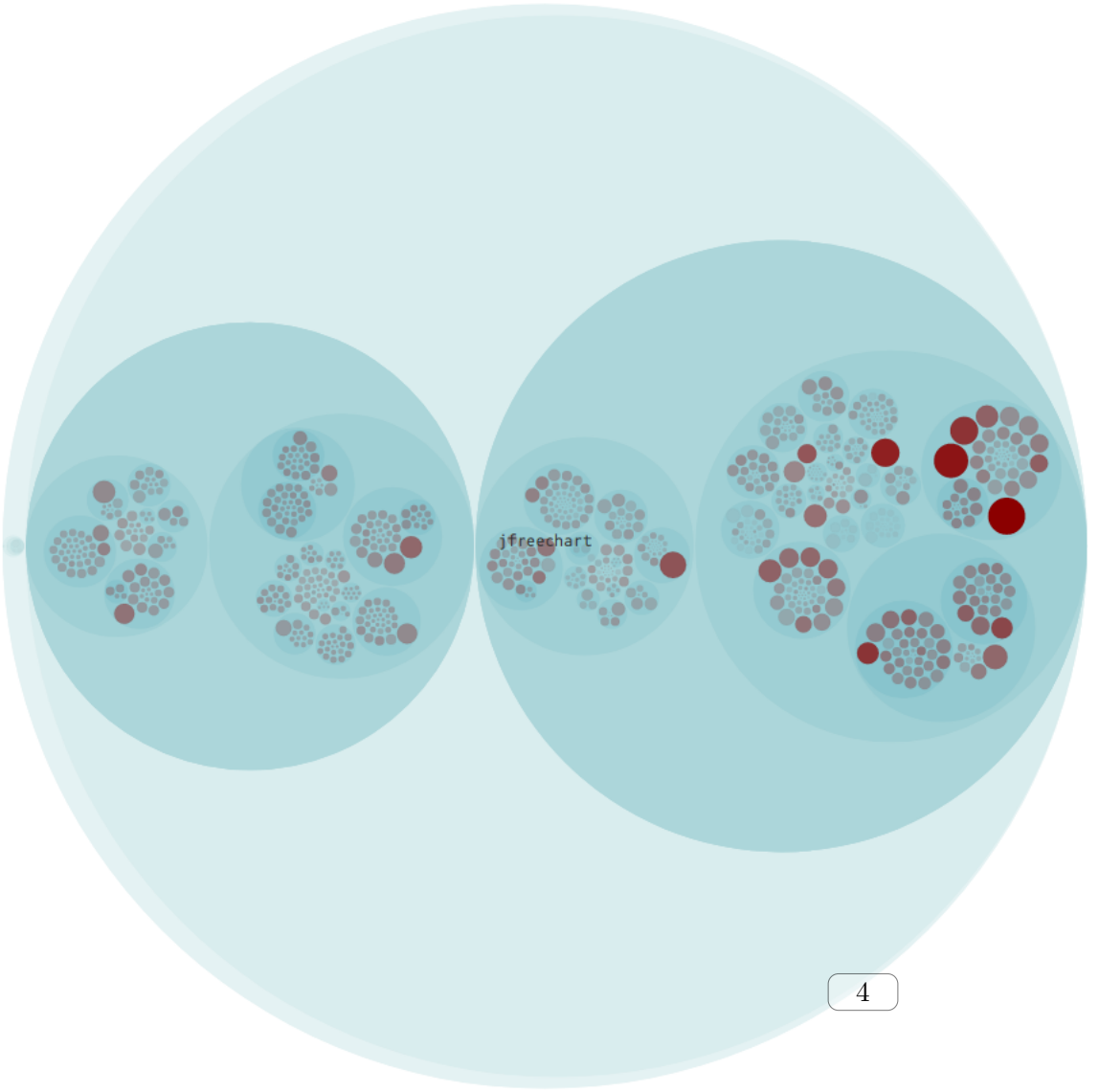
Report generated by Cobertura 1.9.4.1 on 5/17/19 2:56 PM.


XYItemRenderer
CategoryItemRenderer
AbstractRenderer
PolarItemRenderer
AbstractXYItemRenderer
AbstractCategoryItemRenderer
WaferMapRenderer
DefaultPolarItemRenderer
InheritedXYItemRendererClass
The list of classes that actually inherit and fall within our problem domain are listed below:
CandlestickRenderer
HighLowRenderer
SamplingXYLineRenderer
StandardXYItemRenderer
VectorRenderer
WindItemRenderer
XYAreaRenderer
XYAreaRenderer2
XYBarRenderer
XYBlockRenderer
XYBoxAndWhiskersRenderer
XYBubbleRenderer
XYLineAndShapeRenderer
XYDifferenceRenderer
XYDotRenderer
XYShapeRenderer
XYStepAreaRenderer
XYStepRenderer
YIntervalRenderer
InheritedCategoryItemRendererClass
The list of classes that actually inherit and fall within our problem domain are listed below:
AreaRenderer
BarRenderer
BoxAndWhiskerRenderer
CategoryStepRenderer
LevelRenderer
LineAndShapeRenderer
MinMaxCategoryRenderer
ScatterRenderer
InheritedXYItemRendererClassExtenders
Of course, these classes can also have subclasses that also fall in our problem domain.
These include:
ClusteredXYBarRenderer (extends XYBarRenderer)
StackedXYBarRenderer (extends XYBarRenderer)
CyclicXYItemRenderer (extends StandardXYItemRenderer)
DefaultXYItemRenderer (extends XYLineAndShapeRenderer)
DeviationRenderer (extends XYLineAndShapeRenderer)
XYErrorRenderer (extends XYLineAndShapeRenderer)
XYSplineRenderer (extends XYLineAndShapeRenderer)
StackedXYAreaRenderer (extends XYAreaRenderer)
StackedXYAreaRenderer2 (extends XYAreaRenderer2)
InheritedCategoryItemRendererClassExtenders
Of course, these classes can also have subclasses that also fall in our problem domain.
These include:
DefaultCategoryItemRenderer (extends LineAndShapeRenderer)
StatisticalCategoryItemRenderer (extends LineAndShapeRenderer)
IntervalBarRenderer (extends BarRenderer)
StackedBarRenderer (extends BarRenderer)
LayeredBarRenderer (extends BarRenderer)
StatisticalBarRenderer (extends BarRenderer)
WaterfallBarRenderer (extends BarRenderer)
StackedAreaRenderer (extends AreaRenderer)
GroupedStackedBarRenderer (extends StackedBarRenderer)
GhanttRenderer (extends IntervalBarRenderer)


IRendererVisibility
IRendererPaint
IRendererStroke
IRendererShape
IRendererFont
IRendererItem
RenderStateVisibility
RenderStatePaint
RenderStateStroke
RenderStateShape
RenderStateFont
RenderStateItem
XYItemRenderer
CategoryItemRenderer
AbstractRenderer
PolarItemRenderer
AbstractXYItemRenderer
AbstractCategoryItemRenderer
WaferMapRenderer
DefaultPolarItemRenderer
InheritedXYItemRendererClass
The list of classes that actually inherit and fall within our problem domain are listed below:
CandlestickRenderer
HighLowRenderer
SamplingXYLineRenderer
StandardXYItemRenderer
VectorRenderer
WindItemRenderer
XYAreaRenderer
XYAreaRenderer2
XYBarRenderer
XYBlockRenderer
XYBoxAndWhiskersRenderer
XYBubbleRenderer
XYLineAndShapeRenderer
XYDifferenceRenderer
XYDotRenderer
XYShapeRenderer
XYStepAreaRenderer
XYStepRenderer
YIntervalRenderer
InheritedCategoryItemRendererClass
The list of classes that actually inherit and fall within our problem domain are listed below:
AreaRenderer
BarRenderer
BoxAndWhiskerRenderer
CategoryStepRenderer
LevelRenderer
LineAndShapeRenderer
MinMaxCategoryRenderer
ScatterRenderer
InheritedXYItemRendererClassExtenders
Of course, these classes can also have subclasses that also fall in our problem domain.
These include:
ClusteredXYBarRenderer (extends XYBarRenderer)
StackedXYBarRenderer (extends XYBarRenderer)
CyclicXYItemRenderer (extends StandardXYItemRenderer)
DefaultXYItemRenderer (extends XYLineAndShapeRenderer)
DeviationRenderer (extends XYLineAndShapeRenderer)
XYErrorRenderer (extends XYLineAndShapeRenderer)
XYSplineRenderer (extends XYLineAndShapeRenderer)
StackedXYAreaRenderer (extends XYAreaRenderer)
StackedXYAreaRenderer2 (extends XYAreaRenderer2)
InheritedCategoryItemRendererClassExtenders
Of course, these classes can also have subclasses that also fall in our problem domain.
These include:
DefaultCategoryItemRenderer (extends LineAndShapeRenderer)
StatisticalCategoryItemRenderer (extends LineAndShapeRenderer)
IntervalBarRenderer (extends BarRenderer)
StackedBarRenderer (extends BarRenderer)
LayeredBarRenderer (extends BarRenderer)
StatisticalBarRenderer (extends BarRenderer)
WaterfallBarRenderer (extends BarRenderer)
StackedAreaRenderer (extends AreaRenderer)
GroupedStackedBarRenderer (extends StackedBarRenderer)
GhanttRenderer (extends IntervalBarRenderer)

Hotspots identify the modules with most development activity -- often technical debt.

Hotspots | Refactoring Targets | Code Age | Defects | Programming Language

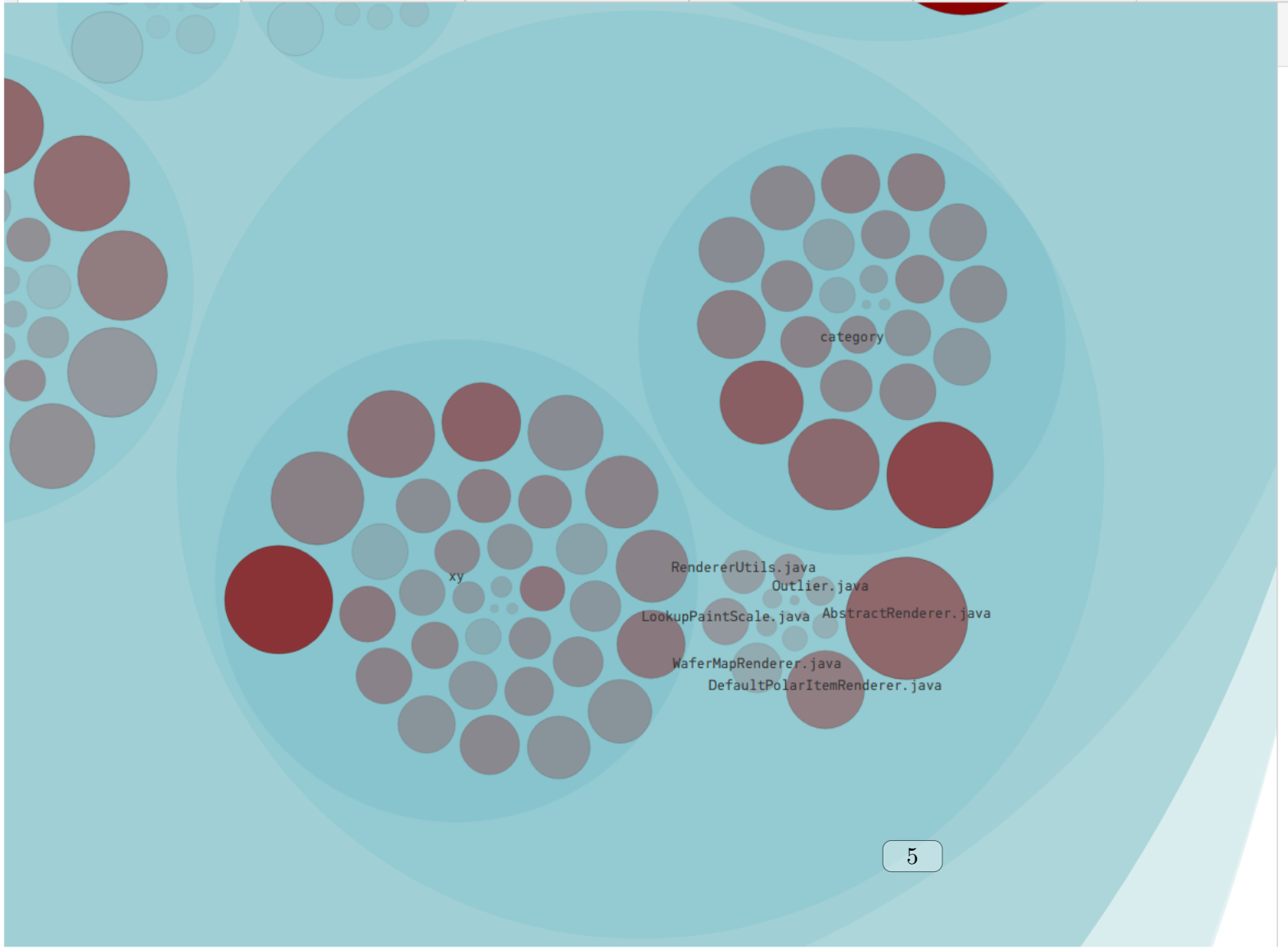


System

jfreechart

Hotspots identify the modules with most development activity -- often technical debt.

Hotspots | Refactoring Targets | Code Age | Defects | Programming Language

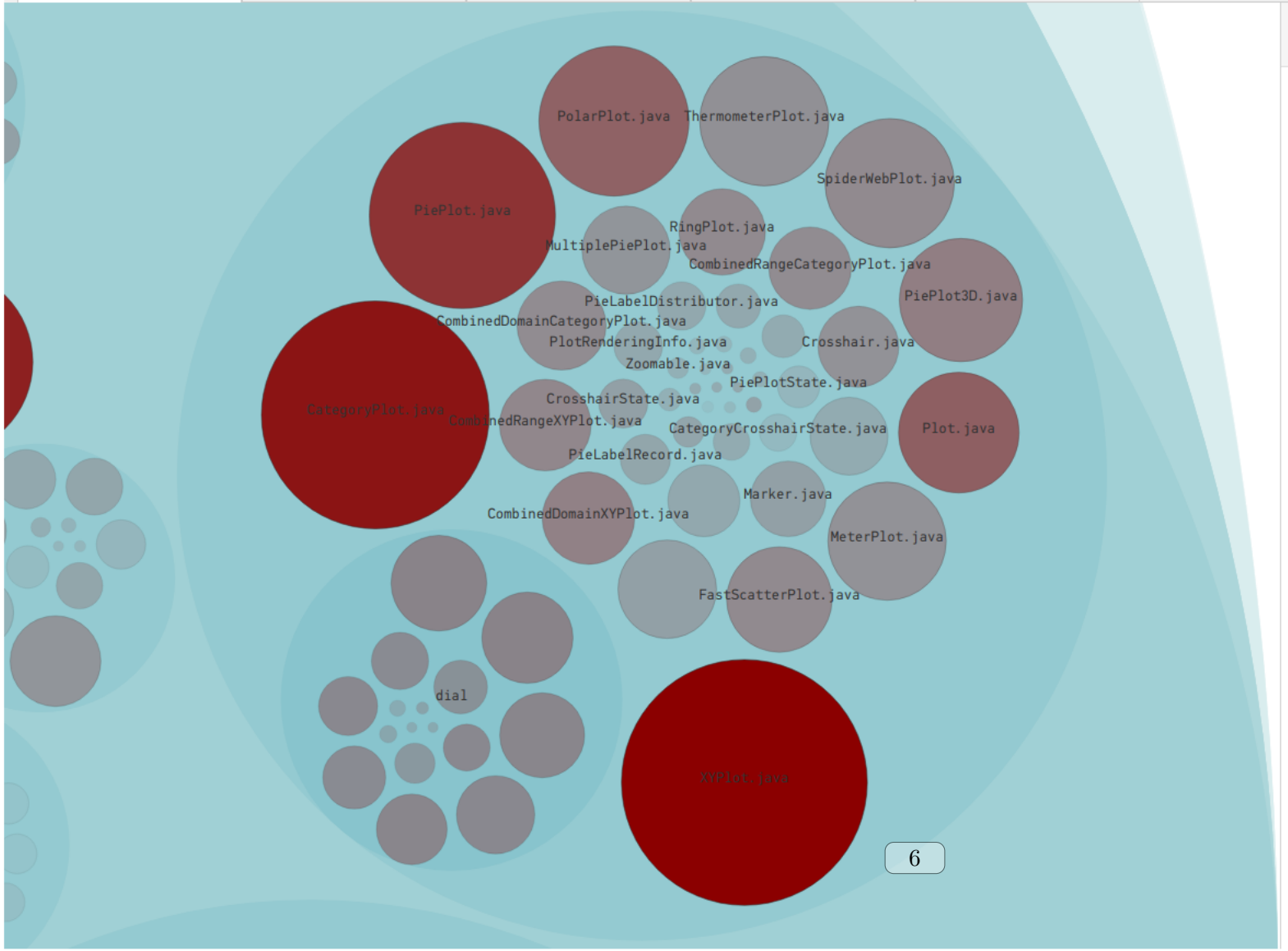


System / jfreechart / src / main / java / org / jfree / chart / renderer

- category
- xy
- AbstractRenderer.java
- AreaRendererEndType.java
- DefaultPolarItemRenderer.java
- GrayPaintScale.java
- LookupPaintScale.java
- NotOutlierException.java
- Outlier.java
- OutlierList.java
- OutlierListCollection.java
- PaintScale.java
- PolarItemRenderer.java
- RendererState.java
- RendererUtils.java
- WaferMapRenderer.java

Hotspots identify the modules with most development activity -- often technical debt.

Hotspots | Refactoring Targets | Code Age | Defects | Programming Language

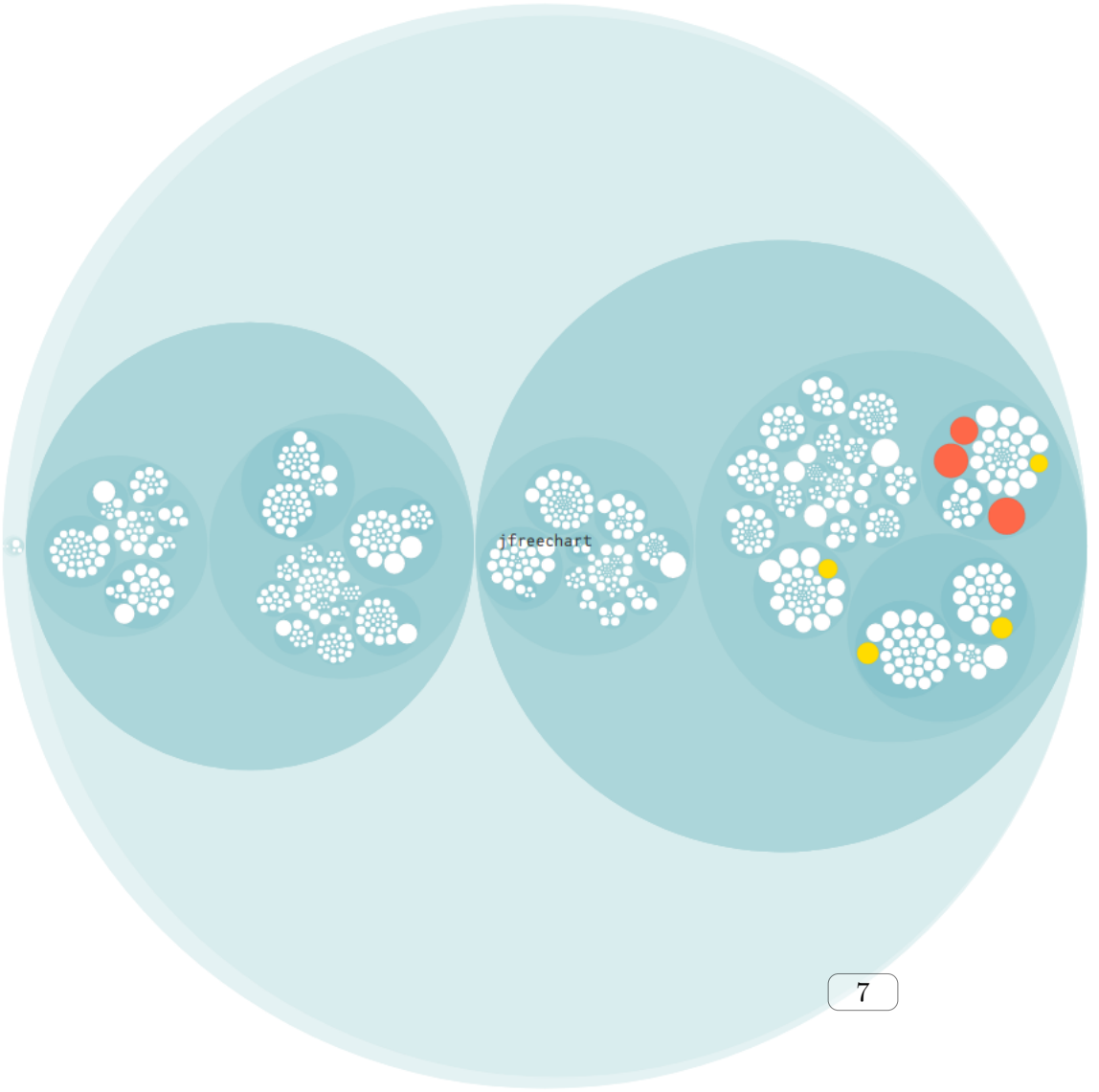


System / jfreechart / src / main / java / org / jfree / chart / plot

- dial
- AbstractPieLabelDistributor.java
- CategoryCrosshairState.java
- CategoryMarker.java
- CategoryPlot.java
- CenterTextMode.java
- CombinedDomainCategoryPlot.java
- CombinedDomainXYPlot.java
- CombinedRangeCategoryPlot.java
- CombinedRangeXYPlot.java
- CompassPlot.java
- Crosshair.java
- CrosshairState.java
- DatasetRenderingOrder.java
- DefaultDrawingSupplier.java
- DialShape.java
- DrawingSupplier.java
- FastScatterPlot.java
- IntervalMarker.java

Prioritize improvements to the highlighted files. Red is most serious.

Hotspots | Refactoring Targets | Code Age | Defects | Programming Language



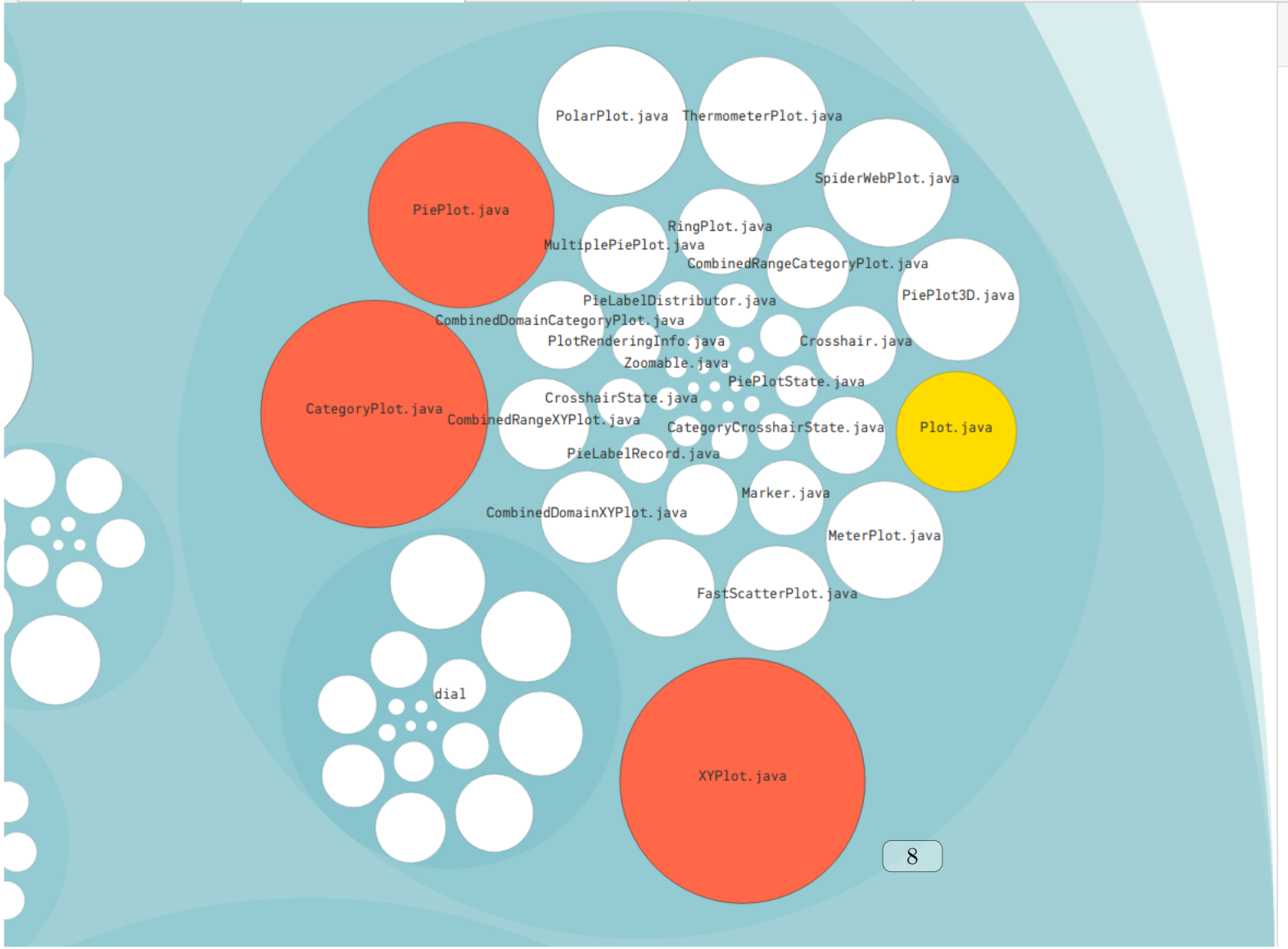
7

System

jfreechart

Prioritize improvements to the highlighted files. Red is most serious.

Hotspots | Refactoring Targets | Code Age | Defects | Programming Language



System / jfreechart / src / main / java / org / jfree / chart / plot

- dial
- AbstractPieLabelDistributor.java
- CategoryCrosshairState.java
- CategoryMarker.java
- CategoryPlot.java
- CenterTextMode.java
- CombinedDomainCategoryPlot.java
- CombinedDomainXYPlot.java
- CombinedRangeCategoryPlot.java
- CombinedRangeXYPlot.java
- CompassPlot.java
- Crosshair.java
- CrosshairState.java
- DatasetRenderingOrder.java
- DefaultDrawingSupplier.java
- DialShape.java
- DrawingSupplier.java
- FastScatterPlot.java
- IntervalMarker.java

Refactoring Targets

Prioritize improvements to these files since they have the highest technical debt interest rate.

jfreechart/src/main/java/org/jfree/chart/plot/XYPlot.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/plot/CategoryPlot.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/plot/PiePlot.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/renderer/xy/AbstractXYItemRenderer.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/renderer/category/AbstractCategoryItemRenderer.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/plot/Plot.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/plot/PolarPlot.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/renderer/xy/XYLineAndShapeRenderer.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/axis/ValueAxis.java	X-Ray
jfreechart/src/test/java/org/jfree/chart/plot/XYPlotTest.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/axis/CategoryAxis.java	X-Ray
jfreechart/src/main/java/org/jfree/chart/renderer/category/BarRenderer.java	X-Ray

X-Ray File Results

Projects / jfreechart / Files / jfreechart/src/main/java/org/jfree/chart/plot/XYPlot.java

Hotspots | Internal Temporal Coupling | Structural Recommendations

Function	Change Frequency	Complexity/Size	Cyclomatic Complexity
equals	7	206	112
draw	6	245	45
drawQuadrants	2	106	24
getDataRange	3	95	20
render	5	83	18
drawRangeGridlines	3	39	13
clone	6	68	12
drawDomainGridlines	2	37	11
getLegendItems	1	30	11
clearDomainMarkers	2	45	10
clearRangeMarkers	2	45	10
removeRangeMarker	5	31	9
readObject	3	55	9
removeDomainMarker	3	29	9
drawAxes	1	73	9
addDomainMarker	4	38	8
setRangeAxis	2	36	8
addRangeMarker	2	35	8
zoomDomainAxes	3	35	7
zoomRangeAxes	3	35	7
setRangeAxisLocation	2	23	7
setDomainAxisLocation	2	23	7
calculateDomainAxisSpace	1	36	7
calculateRangeAxisSpace	1	35	7
checkAxisIndices	1	19	7
drawDomainMarkers	0	23	7
drawRangeMarkers	0	22	7
setDomainAxis	2	23	6
XYPlot	7	96	5
getDatasetsMappedToDomainAxis	4	18	5
getDatasetsMappedToRangeAxis	4	18	5
panDomainAxes	3	15	5
setRenderer	2	23	5
panRangeAxes	2	15	5
getDomainMarkers	0	17	5
getRangeMarkers	0	17	5
removeAnnotation	5	12	4
getDomainAxis	1	14	4
getRangeAxis	1	14	4
getRendererForDataset	1	11	4
getIndexOf	1	9	4
indexOf	1	8 10	4
getQuadrantPaint	1	7	4

X-Ray File Results

Projects / jfreechart / Files / jfreechart/src/main/java/org/jfree/chart/plot/CategoryPlot.java

Hotspots | Internal Temporal Coupling | Structural Recommendations

Function	Change Frequency	Complexity/Size	Cyclomatic Complexity
equals	6	180	101
draw	9	211	34
drawRangeGridlines	4	43	15
render	2	60	13
drawAxes	1	82	13
clone	5	65	12
clearDomainMarkers	1	49	10
clearRangeMarkers	1	49	10
removeRangeMarker	5	29	9
removeDomainMarker	3	28	9
checkAxisIndices	0	24	9
readObject	4	41	8
addDomainMarker	4	36	8
addRangeMarker	2	34	8
calculateDomainAxisSpace	1	43	8
zoomRangeAxes	3	35	7
setDomainAxisLocation	2	23	7
setRangeAxisLocation	2	21	7
calculateRangeAxisSpace	1	33	7
drawDomainGridlines	1	24	7
datasetsMappedToDomainAxis	4	22	6
setRangeAxis	3	23	6
setRenderer	2	26	6
setDomainAxis	2	23	6
getLegendItems	2	17	6
CategoryPlot	9	93	5
datasetsMappedToRangeAxis	3	19	5
getDataRange	1	21	5
panRangeAxes	1	18	5
getRendererForDataset	1	11	5
drawRangeMarkers	0	19	5
drawDomainMarkers	0	19	5
getDomainMarkers	0	17	5
getRangeMarkers	0	17	5
removeAnnotation	5	13	4
getDomainAxisIndex	4	9	4
getDomainAxis	1	14	4
getRangeAxis	1	11	4
getCategoriesForAxis	1	14	4

X-Ray File Results

Projects / jfreechart / Files / jfreechart/src/main/java/org/jfree/chart/plot/PiePlot.java

Hotspots | Internal Temporal Coupling | Structural Recommendations

Function	Change Frequency	Complexity/Size	Cyclomatic Complexity
equals	7	172	99
drawItem	5	83	16
drawPie	5	125	14
drawSimpleLabels	5	82	11
getLegendItems	4	62	10
lookupSectionPaint	5	42	8
clone	3	35	7
drawLabels	4	61	6
draw	4	57	6
lookupSectionOutlineStroke	3	29	6
lookupSectionOutlinePaint	3	29	6
getArcCenter	0	42	6
getMaximumExplodePercent	1	15	5
getSectionKey	0	12	5
drawLeftLabels	7	47	4
drawRightLabels	6	49	4
drawLeftLabel	3	43	4
drawRightLabel	1	44	4
setInteriorGap	1	13	4
getArcBounds	1	20	3
getExplodePercent	1	12 10	3
PiePlot	8	59	2

X-Ray File Results

Projects / jfreechart / Files / jfreechart/src/main/java/org/jfree/chart/renderer/xy/AbstractXYItemRenderer.java

Hotspots | Internal Temporal Coupling | Structural Recommendations

Function	Change Frequency	Complexity/Size	Cyclomatic Complexity
equals	5	45	25
drawRangeMarker	3	157	22
drawDomainMarker	3	157	22
findRangeBounds	4	39	12
getLegendItem	2	51	9
findDomainBounds	3	21	8
getLegendItems	0	21	7
addEntity	3	28	5
updateCrosshairValues	4	27	4
addAnnotation	4	21	4
drawDomainLine	3	28	4
drawAnnotations	2	22	4
drawRangeLine	1	27	4
calculateRangeMarkerTextAnchorPoint	1	17	3

X-Ray File Results

Projects / jfreechart / Files / jfreechart/src/main/java/org/jfree/chart/renderer/category/AbstractCategoryItemRenderer.java

Hotspots | Internal Temporal Coupling | Structural Recommendations

Function	Change Frequency	Complexity/Size	Cyclomatic Complexity
equals	4	47	23
drawRangeMarker	4	163	22
clone	4	71	16
getLegendItems	2	33	12
drawDomainMarker	3	72	9
getLegendItem	1	44	9
findRangeBounds	3	23	8
addEntity	1	31	6
addItemEntity	3	20	4
initialise	2	29	4
drawDomainGridline	1	32	4
drawRangeLine	1	27	4
calculateRangeMarkerTextAnchorPoint	1	16	3
calculateDomainMarkerTextAnchorPoint	1	15	3