FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Extension of the JFreeChart library with Market Profile charts
Student:	Beksultan Baatyrbekov
Supervisor:	Ing. Jan Trdlička, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

1) Familiarize yourself with the Java chart library JFreeChart.
2) Study what a Market Profile is, how to calculate it and plot it as a graph.
3) Find out if there are any libraries that can plot Market Profile charts and compare their properties.
4) Design and implement an extension to the JFreeChart library that allows the calculation and rendering of Market Profile graphs.
5) Create a demo application that will illustrate the use of your extension and verify its functionality.
6) Compare the features of your solution with existing libraries/frameworks.

Electronically approved by Ing. Michal Valenta, Ph.D. on 12 October 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Extension of the JFreeChart library with Market Profile charts

Beksultan Baatyrbekov

Department of Software Engineering
Supervisor: Ing. Jan Trdlička, Ph.D.

March 21, 2022

Acknowledgements

I would like to thank my supervisor, Ing. Jan Trdlička, Ph.D., for his support and guidance during this project.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on March 21, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Baatyrbekov, Beksultan. *Extension of the JFreeChart library with Market Profile charts.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce se zaměřuje na analýzu knihovny JFreeChart Java Chart a konkrétního typu finančních grafů – Market Profile Charts a cílem této práce je rozšířit knihovnu JFreeChart o možnost vykreslování Market Profile Charts. Vytvořené řešení umožňuje vykreslovat grafy tržního profilu s objemovými profily.

Klíčová slova JFreeChart Library, Market Profile Schéma, Java, OHLC Dataset

Abstract

This thesis focuses on analyzing JFreeChart Java Chart library and specific type of financial charts–Market Profile Charts and the goal of this thesis is to extend JFreeChart library to enable rendering Market Profile Charts. Created solution allows to render Market Profile Charts with Volume Profiles.

Keywords JFreeChart Library, Market Profile Chart, Java, OHLC Dataset

Contents

List of Figures

List of Tables

Introduction

It is all about having the correct tools when it comes to investing. If an individual does not have the right resources, one will not be able to make informed trading decisions, whether one is trying to monitor volatility measurements or interpret price action. The market profile chart is one such resource. The market profile organizes diverse trading data so that traders and investors may make informed trading decisions, particularly it may offer a unique prospective on buying and selling opportunities.
However, learning how to utilize them takes time and effort, and small number of stock chart services provide charting of market profile. Significant drawback when it comes to use existing charting tools for plotting market profile is huge cost to obtain software license or subscription.
To provide more variety among software that allows to plot market profile, this thesis introduces an extension to free charting library "JFreeChart" with rendering market profile charts. Extended JFreeChart library with Market Profile allows developers to add functionality of charting Market Profile in software applications or create standalone software with Java.

Goal of the thesis

The goal of the thesis is to extend JFreeChart Charting library with market profile charts.

- Analyse JFreeChart library
- Analyse market profile charts
- Design an implement solution that allows plotting market profile charts with JFreeChart
- Create a demo application that will illustrate extended library with market profile charts

- Compare the features of provided solution with existing libraries/frameworks

Chapter 1

Analysis

In this chapter, the analysis of market profile charts and JFreeChart library will be performed. The chapter will hellp to understand market profile charts and what JFreeChart offers and what JFreeChart does not offer.

1.1 Market Profile

"A Market Profile is an intra-day charting technique (price vertical, time/activity horizontal) devised by J. Peter Steidlmayer, a trader at the Chicago Board of Trade (CBOT), ca 1959-1985. Steidlmayer was seeking a way to determine and to evaluate market value as it developed in the day time frame. The concept was to display price on a vertical axis against time on the horizontal, and the ensuing graphic generally is a bell shape–fatter at the middle prices, with activity trailing off and volume diminished at the extreme higher and lower prices." [1]

1.1.1 Introduction to Market Profile

Market Profile was introduced in 1980s by the Chicago Board Of Trade (CBOT) director, Peter Steidlmayer. Peter's vision about the concept of Market Profile was to provide more transparency for off-floor trader, which would bring in new traders into trading. Furthermore, opening markets to a larger audience would help the financial markets flourish and generate more money and prosperity. On the contrary many on-floor traders worried about their livelihood were resistant to this idea, believing that market data and floor information should be a privilege to individuals who traded on the floor. Peter Steidlmayer developed a system that displays trading data from the market and depicts activity occurring during the trading session using the statistical bell curve and published CBOT Market Profile Guide "A Six-Part Study Guide To Market Profile" in 1985. The Market Profile has evolved since then, but the fundamental ideas have remained the same. [2]

1.1.2 What are Market Profile Charts?

Market Profile charts combine price and time to create a unique method to visualize market behaviour and the day's most crucial prices. A different letter assigned to each defined time period of every trading session. Because the Market profile captures both price and time, it allows us to see how the market has changed over time. One of the biggest benefits of a market profile is that it can be utilized in any market you want, including Forex, Futures, Cryptocurrencies, and Stocks. To display the data, the market profile uses TPO Charts.

1.1.3 Key Terminology In Market Profile

1.1.3.1 Value Area

The value area in the Market profile comprises 70% of the data within one standard deviation of the mean. To put it another way, it's the area that contains 70% of all trades made in a single session. We may infer that buyers and sellers agreed on prices in this area because here is where the majority of the activity occurred during the session.

- **Value Area High (VAH)** – The upper level of value area.
- **Value Area Low (VAL)** – The lower level of value area.

In the center of the Value Area, we can find the Point of Control (POC).

1.1.3.2 Point of Control (POC)

The point of control is a specific price where the most volume has been executed at the price that is closest to the center of the value area. The point of control acts as a key pricing magnet. The greater the number of the Time Price Opportunities (TPOs) that make up the Point Of Control (POC), the more significant it becomes.

1.1.3.3 Time Price Opportunity (TPO)

This is the most fundamental component of a market profile chart. Each time the asset traded at a specific price during the trading day, this is referred to as a time-price opportunity. TPO is usually represented by letters on a market profile chart, and intervals may be 1-minute, 5-minute, 30-minute etc. For example, the letter "A" may stand for the first minute of trading, the letter "B" for the next minute, and so on. Lower case lettered time blocks represent out-of-market hours trading. It's worth mentioning that some charts utilize colors rather than letters to indicate time increments. See example of a table with time blocks (Fig. 1.1).

Market Profile Time Block	Start Time	End Time
A — Market Open	08:00:00	08:01
B	08:01	08:02
C	08:02	08:03
D	08:03	08:04
E	08:04	08:05
F	08:05	08:06
G	08:06	08:07
H	08:07	08:08
I	08:08	08:09
J	08:09	08:10
K	08:10	08:11
L	08:11	08:12
M	08:12	08:13
N	08:13	08:14
O	08:14	08:15

Table 1.1: Examlpe of Time Blocks

1.1.3.4 Initial Balance

- **Range** – The range of a day's price action from high to low.
- **Range Extension** – An extension of price beyond the initial balance.

The Initial Balance (IB) is the price range in which the market spent its first two TPO sessions after opening. To put it another way, the initial balance is the range of the market's first hour after it opens. Outside of the initial balance, we can observe range extensions to get a perspective of who is in control.

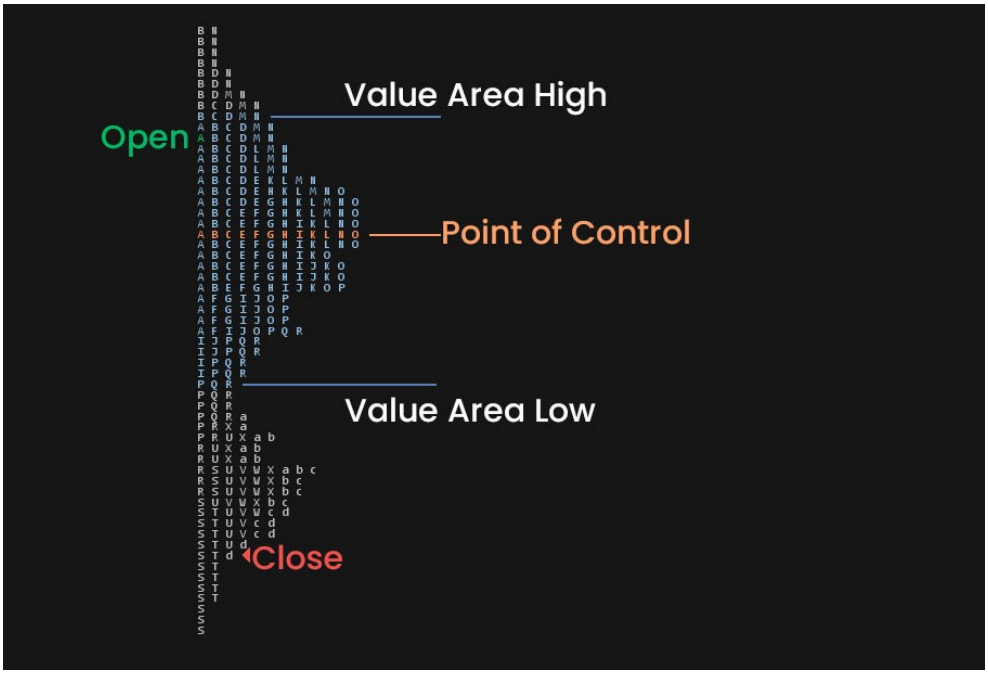

Figure 1.1: Example of Market Profile Chart wrt Terminology [3]

1.1.4 Market Profile vs Volume Profile

Although many places and people would use the terms interchangeably, Market profile and Volume profile are two different things. Volume profiles show how much volume was done at prices, while Market Profile or TPO (time-price-opportunity) profiles show how much time was spent at prices. The majority of profile traders utilize both in combination, although they favor one over the other.

1.2 JFreeChart

1.2.1 Overview

JFreeChart is a free Java class library that allows programmers generate a wide range of graphs and charts for us both in client and server side applications.

The JFreeChart project was started in February 2000 and continues to be managed by David Gilbert with contributions from a diverse community of developers. "Today, JFreeChart is the most widely used chart library for Java (see a list of some of the products and projects that use JFreeChart)" [4]. The library is distributed with a complete source code subject to the terms of GNU Lesser General Public License, which permits JFreeChart to be used in proprietary or free software applications.

"JFreeChart is used to generate widely used chart types, including pie charts, bar charts (regular and stacked, with an optional 3D-effect), histograms, time series charts (including moving averages, high-low-open-close charts and candlestick plots), Gantt charts meter charts (dial, compass and thermometer), symbol charts and other types of charts." [5]

JFreeChart can be used on the client-side of applications as well as server on the server-side of applications. For Java Swing-based applications, JFreeChart is a great solution. The Data set object, which stores the data to be displayed in the charts, is the library's basic data structure. Many distinct Data set objects are available in JFreeChart, all of which implement the Data set interface and may be used to construct various sorts of charts, such as XYBarDataset (Bar plot) and DefaultPieDataset (Pie chart). Zooming, labels, colors, and tool tips can all be customized on the chart. JFreeChart's key advantages are its excellent documentation, example code, minimal dependencies, and flexible customizability.

It can also export to a variety of popular formats, including JPEG, PNG, and PDF, and it may be used in JSP[1]/servlet[2]-based applications to dynamically stream charts to web pages. Orson Charts, a Java 3D chart library that can generate a wide variety of 3D charts in client-side (Java FX and Java Swing) and server-side applications, was recently released by the JFreeChart development team.

[1] Java Server Pages (JSP) – a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

[2] Servlet – a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

Latest version of JFreeChart at the moment of writing this thesis, 1.5.3 (see at jfreechart github page), was released in February 2021. 1.5.3 version of JFreeChart requires JDK 8 or later. If JavaFX support is required, there is also need to include the JFreeChart-FX extensions (see at jfreechart javafx extension github page).
JFreeChart has two versions of documentation:

- a free version, the "JFreeChart Installation Guide", is available from the JFreeChart home page, and contains chapters up to and including the instructions for installing JFreeChart and running the demo;
- a premium version, the "JFreeChart Developer Guide", is available only to those that have paid for it, and includes additional tutorial chapters and reference documentation for the JFreeChart classes

1.2.2 Creating chart with JFreeChart

Creating charts with JFreechart is a three step process. The following steps are needed to follow to create charts:

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

1.2.3 Data

A design principle in JFreeChart is that there should be a clear separation between the data and it's presentation. To create Market Profile Chart we need a dataset that supplies data in the form of open-high-low-close items. Which relates to trading data (prices or rates) in financial markets: the open and close values represent the prices at the opening and closing of the trading period, while the high and low values represent the highest and lowest price during the trading period. Another useful field needed is volume, which represents the volume of trading. The data described above can be represented by OHLCDataset Interface in JFreeChart.

1.2.3.1 OHLCDataset Interface

As previously stated, a dataset that supplies data in the form of open-high-low-close items and volume. This interface is an extension of the XYDataset interface. Interface Methods:

- public double getHighValue(int series, int item) – Returns the high value for an item in a series.

- public double getLowValue(int series, int item) – Returns the low value for an item in a series.
- public double getOpenValue(int series, int item) – Returns the open value for an item in a series.
- public double getCloseValue(int series, int item) – Returns the close value for an item in a series.
- public double getVolumeValue(int series, int item) – Returns the trading volume for an item in a series, or Double.NaN if no trading volume isrecorded.

This interface is implemented by several classes:

- DefaultHighLowDataset
- DefaultOHLCDataset
- OHLCSeriesCollection

OHLCDataset is used to plot charts by the CandlestickRenderer and HighLowRenderer, provided by JFreeChart and is used by MarketProfileRenderer in my implementation.

1.2.4 JFreeChart class

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

- public void draw(Graphics2D g2, Rectangle2D area) – instructs the JFreeChart object to draw a chart onto a specific area on some graphics device.

Java supports several graphics devices—including the screen, the printer, and buffered images—via different implementations of the abstract class java.awt.Graphics2D. Thanks to this abstraction, JFreeChart can generate charts on any of these target devices, as well as others implemented by third parties (for example, the SVG Generator implemented by the Batik Project).
In broad terms, the JFreeChart class sets up a context for drawing a Plot. The plot obtains data from a Dataset, and may delegate the drawing of individual data items to a CategoryItemRenderer or an XYItemRenderer, depending on the plot type (not all plot types use renderers).
The most important method for a chart is the draw() method:

- public void draw(Graphics2D g2, Rectangle2D area) – 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 area) – 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).

1.2.5 Plot

An abstract base class that controls the visual representation of data in a chart. The JFreeChart class maintains a reference to a Plot, and will provide it with an area in which to draw itself (after allocating space for the chart titles and legend).
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.

1.2.6 XYPlot

Draws a visual representation of data from an XYDataset, where the domain axis measures the x-values and the range axis measures the y-values.
The type of plot is typically displayed using a vertical orientation, but it is possible to change to a horizontal orientation which can be useful for certain applications.

1.2.6.1 Layout

Axes are laid out at the left and bottom of the drawing area. The space allocated for the axes is determined automatically. The following diagram (fig 1.2) 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).

1.2.6.2 Datasets and Renderers

An XYPlot can have zero, one or many datasets and each dataset is usually associated with a renderer (the object that is responsible for drawing the visual

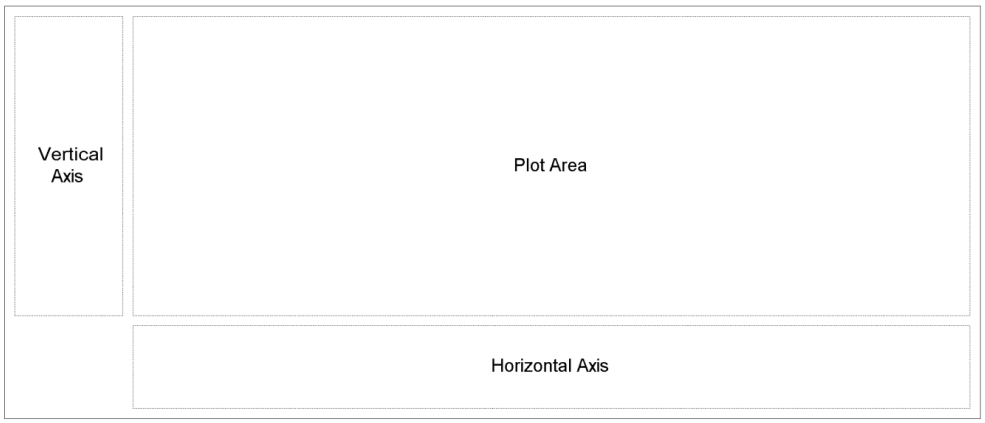

Figure 1.2: The plot layout

representation of each item in a dataset). A dataset is an instance of any class that implements the XYDataset interface and a renderer is an instance of any class that implements the XYItemRenderer interface.

1.2.7 Financial Charts

JFreeChart library at the moment has a functionality of rendering popular financial charts: (1) Candlestick Charts and (2) High Low Charts.

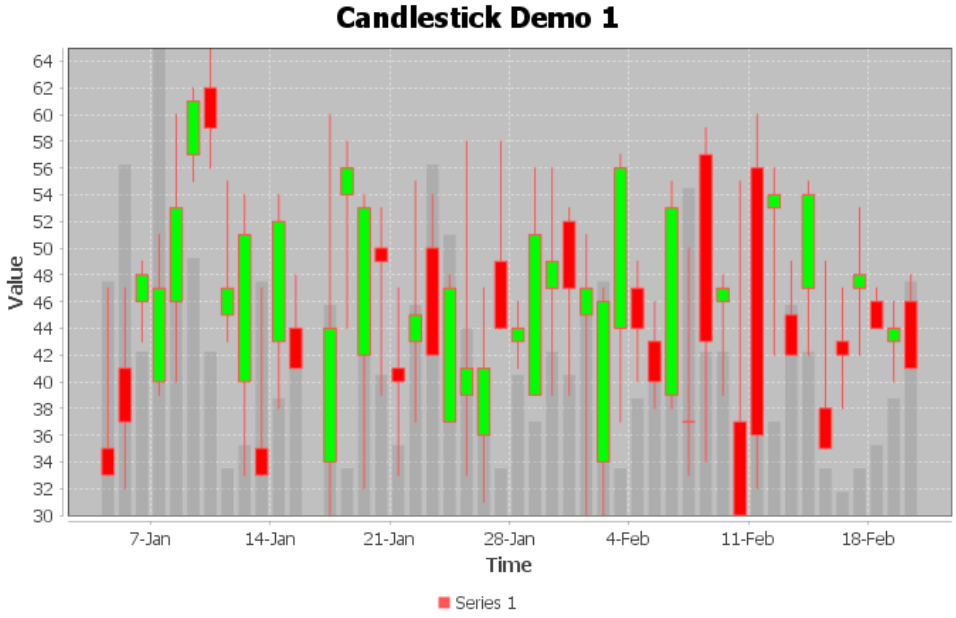

Figure 1.3: Candlestick Chart Demo with JFreeChart library

"A candlestick chart (also called Japanese candlestick chart) is a style of financial chart used to describe price movements of a security, derivative, or currency." [6] Candlestick charts visualize specific number of candles across time and prices, where each candle represent four dimensions of price in a trading period[3] and these four dimensions are the open, the high, the low and the close.

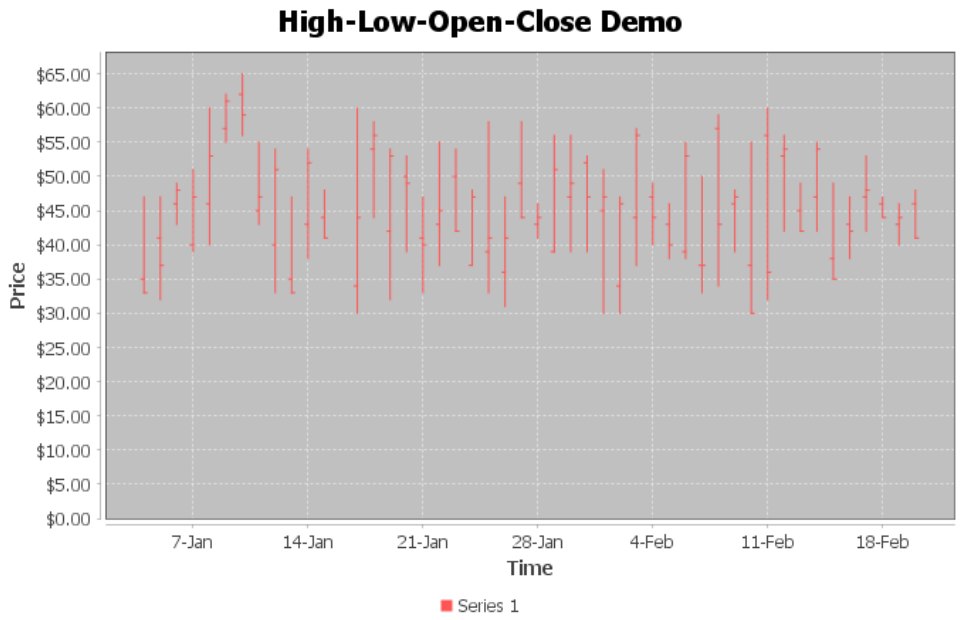

Figure 1.4: OHLC Chart with JFreeChart library

"Open-high-low-close Charts (or OHLC Charts) are used as a trading tool to visualise and analyse the price changes over time for securities, currencies, stocks, bonds, commodities, etc."[7] The y-axis on an OHLC Chart is used for the price scale, while the x-axis is the timescale. On each single time period, an OHLC Charts plots a symbol that represents two ranges: the highest and lowest prices traded, and also the opening and closing price on that single time period (for example in a day). On the range symbol, the high and low price ranges are represented by the length of the main vertical line. The open and close prices are represented by the vertical positioning of tick-marks that appear on the left (representing the open price) and on right (representing the close price) sides of the high-low vertical line.

[3] A trading period – a time period from one second upwards.

1.3 Requirements

1.3.1 Market Profile renderer

Following requirements need to be met to achieve the goal of rendering market profile charts with JFreeChart library:

- Plot a data item for a provided dataset, where each data item contains: date and time, open price, high price, low price, close price and optionally, volume amount.
- Calculate the domain-axis (x-axis) and the range-axis (y-axis) values for each data point in the dataset to draw it on a chart.

1.3.2 Demo application

Following requirements need to be met to achieve the goal of demonstration of drawing market profile charts with JFreeChart library:

- Read date, time and open-high-low-close data from a csv file.
- Plot candlestick chart from provided dataset.
- Plot market profile chart from provided dataset.
- Plot combined charts with candlestick and market profile charts.

Chapter 2

Design

2.1 Technologies

This section contains the technologies used during the development process and why exactly these technologies are chosen.
During the development, I used IntelliJ IDEA – an integrated development environment (IDE) written in Java for developing computer software. The IDE provides certain features like code completion by analyzing the context, code navigation which allows jumping to a class or declaration in the code directly, code refactoring, code debugging, linting and options to fix inconsistencies via suggestions.
As for the developement of extension of JFreeChart Java library, I chose latest JFreeChart library released–version 1.5.3 at the moment of writing this thesis work. The JFreeChart 1.5.3 version requires JDK(The Java Development Kit) 8 or later to support existing features of the library. To develop a solution for the assignment I have chosen Java 8, because this version of java is widely used among professional and non-professional developers. Unit-tests were performed with junit and powermockito libraries.

2.1.1 Java

"Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere, meaning that compiled Java code can run on all platforms that support Java without the need to recompile." [8]

2.1.2 JDK

"The Java Development Kit (JDK) is a distribution of Java Technology by Oracle Corporation. It implements the Java Language Specification (JLS)

and the Java Virtual Machine Specification (JVMS) and provides the Standard Edition (SE) of the Java Application Programming Interface (API). It is derivative of the community driven OpenJDK which Oracle steward." [9]

2.1.3 JFreeChart

The previous generation of versions of JFreeChart library is 1.0.x generation of version. It required JDK 1.6.0 or higher versions. The JFreeChart version 1.5.3 was released on February 21, 2021. It requires JDK 8 or higher. JFreeChart provides wide variety of demo applications with latest released version of the library. The version 1.5.3 was chosen, because it supports JDK 8 and it is the latest released version at the moment of writing this thesis.

2.1.4 OpenCSV

To read OHLC(open-high-low-close) data from csv file, opencsv Java library was used in the demo application. Opencsv is an easy-to-use CSV (comma-separated values) parser library for Java. [10] It was developed because all the CSV parsers at the time didn't have commercial-friendly licenses. Java 8 is currently the minimum supported version. Opencsv supports all the fundamental CSV-type functionalities, like reading arbitrary numbers of values per line, ignoring commas in quoted elemments, configurable separator and quote characters.

2.1.5 Comma-separated values (CSV)

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. A CSV file typically stores tabular data (numbers and text) in plain text, in which case each line will have the same number of fields. [11] The csv file format for dataset was used, because many resources provide OHLC data in csv format and most importantly it is easy to read file format for datasets.

2.1.6 JUnit

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. [12]

2.1.7 PowerMockito

PowerMock is a framework that extends other mock libraries such as EasyMock with more powerful capabilities. PowerMock uses a custom classloader and bytecode manipulation to enable mocking of static methods, constructors, final classes and methods, private methods, removal of static initializers and more. By using a custom classloader no changes need to be done to the IDE or continuous integration servers which simplifies adoption. Developers familiar with the supported mock frameworks will find PowerMock easy to use, since the entire expectation API is the same, both for static methods and constructors. PowerMock aims to extend the existing API's with a small number of methods and annotations to enable the extra features. [13]

2.2 Dataset class for MarketProfile Renderer

MarketProfileRenderer requires a dataset that contains high, low prices and volume of the trading period. OHLCDataset Interface a dataset that supplies data in the form of open-high-low-close items and volume. The dataset is an extension of XYDataset interface, which is a given argument in the main methods initialise and drawItem inherited from XYItemRenderer.

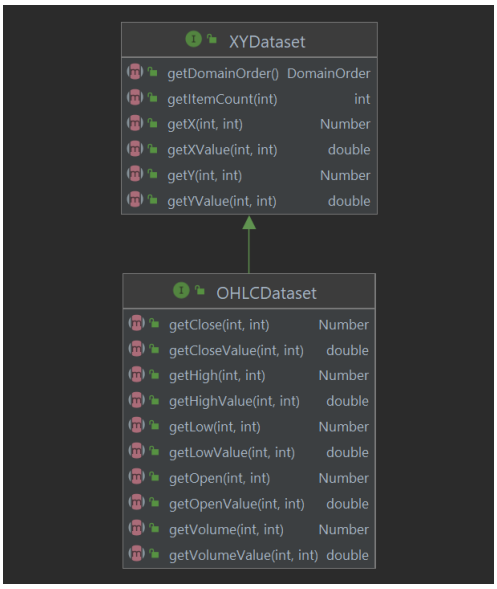

Figure 2.1: OHLCDtaset and XYDataset diagram

2.3 MarketProfileRenderer class model

As previously mentioned, the JFreeChart class coordinates the entire process of drawing charts and sets up a context for drawing a Plot. For XY Plots the XYPlot obtains data from a XYDataset and delegates the drawing of individual data items to XYItemRenderer.
MarketProfileRenderer extends AbstractXYItemRenderer and implements XYItemRenderer interface to draw a visual representation of data from an XYDataset onto plot area.
The class calculates domain-axis (x-axis) values from date and time values and range-axis (y-axis) values from high and low values of the dataset.
The class saves calculated state of the data in ChartItem objects and the data is used to draw items on a chart.

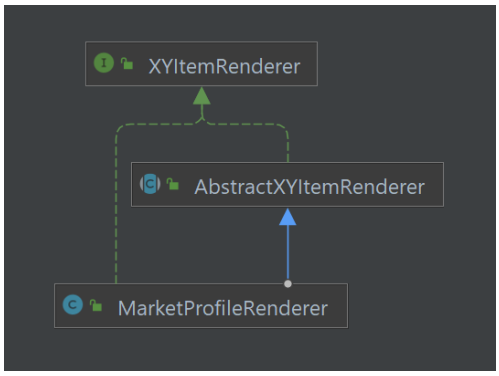

Figure 2.2: Market Profile Renderer Diagram

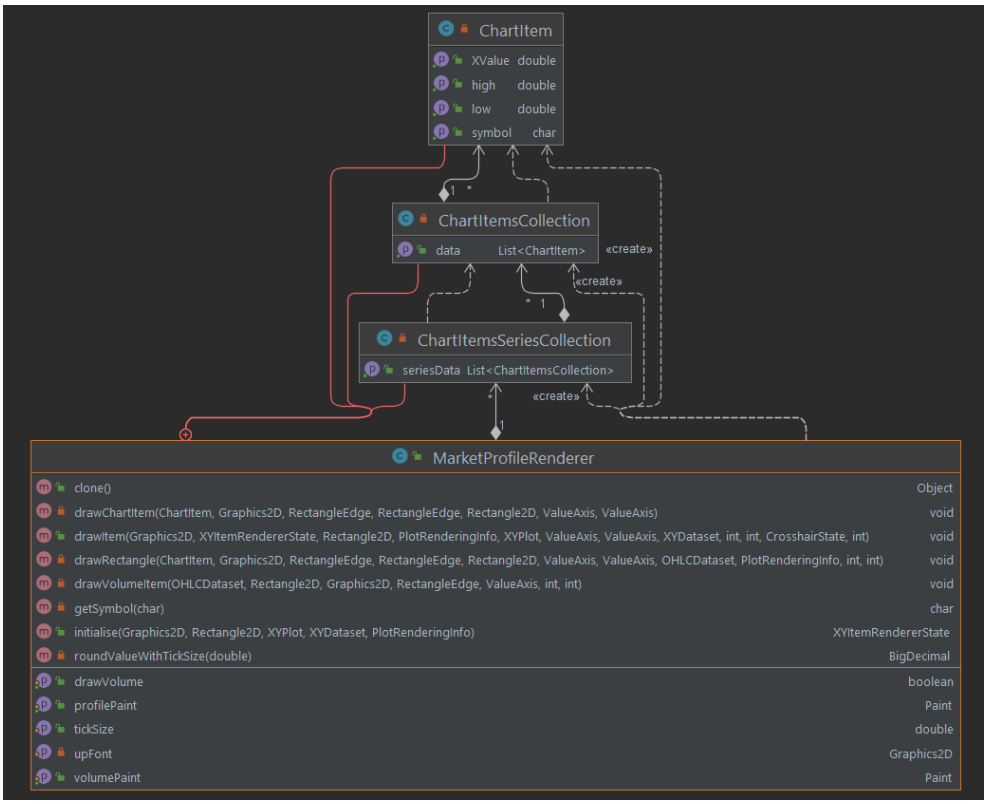

Figure 2.3: MarketProfileRenderer Class Diagram

Chapter 3

Implementation

In this chapter, implementation details of the solution are discussed.

3.1 MarketProfileRendnerer Implementation

This section goes into detail regarding the implementation of MarketProfileRenderer.

3.1.1 Structure

The code structure of the MarketProfileRenderer is implemented in a single class to follow existing structure of JFreeChart library renderers. All implementations of XYItemRenderer interface is implemented in a single class to be used by XYPlot class later to draw corresponding data items on a chart. The most important methods of the class are:

- initialise() – method is called once at the beginning of the chart drawing process and pre-calculates values for domain-axis and range-axis.
- drawItem() – method is responsible for drawing each data item on the chart and uses pre-calculated values for domain-axis and range-axis.

3.1.2 Initialization

The initialise() method is called once at the beginning of the chart drawing process. The method gets the price tick size [4] from the plot object provided in the arguments of the method. The method calculates difference between high and low prices of a data item and calculates their respective values on domain axis(x-axis). Each data item's price represent its plot value on range axis (y-axis). A map with key as a price and value as a x-value used to temporarily

[4] A tick is a measure of the minimum upward or downward movement in the price of a security

store each data item to calculate the next data item. Current data item's x-value is previous data item's x-value added with time step if their prices match. In case the price do not match with none of the previous data item's price, then it's x-value is set to lowerbound x-value. Their respective letter is calculated with their actual time value, where each data item's time is increased linearly, their respective letter is increased. If the method runs out of letters from upper case symbols to lower case symbols, then the letter-system is reset. After data each data item is split into smaller subset of items with tick size and the method calculates their respective x-values and assigns corresponding letters, a list of ChartItems created. The list of ChartItems represent one data item from the dataset, however it is split into smaller items with tick size, and therefore it is stored as list of ChartItems. Each data item is iterated with their series id and item id given by dataset. The method saves the list of ChartItems in ChartItemsCollection, where ChartItemsCollection represent one data item from the dataset. ChartItemsCollection then saved in the ChartItemsSeriesCollection, which represents one series from dataset. ChartItemsSeriesCollection is stored in the list chartItemsSeriesCollections as a field of the MarkteProfileRenderer, where it's index in the list is the series id.

```
@Override
public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset xyDataset, PlotRenderingInfo info) {
    if(!tickSizeConst) {...}

    DateAxis dateAxis = (DateAxis)plot.getDomainAxis();
    domainTickSize = dateAxis.getTickUnit().getSize();

    chartItemsSeriesCollections = new ArrayList<>();
    Map<BigDecimal, Double> xValsMap = new HashMap<>();
    char symbol = '#';
    OHLCDataset dataset = (OHLCDataset) xyDataset;

    for(int series = 0; series < dataset.getSeriesCount(); series ++) {
        List<ChartItemsCollection> chartItemsCollections = new ArrayList<>();
        for(int item = 0; item < dataset.getItemCount(series); item++) {

            symbol = getSymbol(symbol);


            double low = dataset.getLowValue(series, item);
            BigDecimal lowBD = roundValueWithTickSize(low);

            double high = dataset.getHighValue(series, item);
            BigDecimal highBD = roundValueWithTickSize(high);

            List<ChartItem> chartItemsList = new ArrayList<>();

            while (highBD.compareTo(lowBD) > 0) {...}
            chartItemsCollections.add(new ChartItemsCollection(chartItemsList));


            if(drawVolume) {...}
        }
        chartItemsSeriesCollections.add(new ChartItemsSeriesCollection(chartItemsCollections));
    }
    return new XYItemRendererState(info);
}
```

Figure 3.1: initialise() method

3.1.3 Drawing

Method drawItem is called for each data item to draw item onto plot. The method looks for the list chart items that represent one data item and delegates actual drawing to drawChartItem and drawRectangle methods. In the drawChartItem and drawRectangle methods, x and y values of the chart item are translated into Java 2D values with rangeAxis and domainAxis objects. DrawRectangle draws one rectangle in the chart, which represents a time-price opportunity and drawChartItem method draws one letter inside of the corresponding rectangle. Then finally Graphics2D draws items onto plot.

```
@Override
public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
                     PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
                     XYDataset xyDataset, int series, int item, CrosshairState crosshairState, int pass)
{
    RectangleEdge rangeEdge = plot.getRangeAxisEdge();
    RectangleEdge domainEdge = plot.getDomainAxisEdge();
    OHLCDataset dataset = (OHLCDataset) xyDataset;

    List<ChartItem> chartItemList = chartItemsSeriesCollections.get(series).getSeriesData().get(item).getData();
    for(ChartItem chartItem : chartItemList) {
        drawRectangle(chartItem, g2, domainEdge, rangeEdge, dataArea, domainAxis, rangeAxis, dataset, info, series, item);
        drawChartItem(chartItem, g2, domainEdge, rangeEdge, dataArea, domainAxis, rangeAxis);
    }

    if(drawVolume) {
        drawVolumeItem(dataset, dataArea, g2, domainEdge, domainAxis, series, item);
    }
}
```

Figure 3.2: drawItem() method

```
private void drawChartItem(ChartItem chartItem, Graphics2D g2, RectangleEdge domainEdge, RectangleEdge rangeEdge,
                           Rectangle2D dataArea, ValueAxis domainAxis, ValueAxis rangeAxis)
{
    setUpFont(g2);

    double y = (chartItem.getLow() + chartItem.getHigh())/2;
    double x = (chartItem.getXValue() + chartItem.getXValue() + domainTickSize)/2;
    String text = Character.toString(chartItem.getSymbol());

    double yJ2D = rangeAxis.valueToJava2D(y, dataArea, rangeEdge);
    double xJ2D = domainAxis.valueToJava2D(x, dataArea, domainEdge);

    g2.drawString(text, (float) xJ2D, (float) yJ2D);
}
```

Figure 3.3: drawChartItem() method

```
private void drawRectangle( ChartItem chartItem, Graphics2D g2, RectangleEdge domainEdge, RectangleEdge rangeEdge,
                            Rectangle2D dataArea, ValueAxis domainAxis, ValueAxis rangeAxis,
                            OHLCDataset dataset, PlotRenderingInfo info, int series, int item)
{
    double lowJ2D = rangeAxis.valueToJava2D(chartItem.getLow(), dataArea, rangeEdge);
    double highJ2D = rangeAxis.valueToJava2D(chartItem.getHigh(), dataArea, rangeEdge);

    double x1J2D = domainAxis.valueToJava2D(chartItem.getXValue(), dataArea, domainEdge);
    double x2J2D = domainAxis.lengthToJava2D(domainTickSize, dataArea, domainEdge);

    Rectangle2D.Double body = new Rectangle2D.Double(x1J2D, highJ2D, x2J2D,  h: lowJ2D - highJ2D);
    Rectangle2D.Double hotspot = new Rectangle2D.Double(x1J2D, highJ2D, x2J2D,  h: lowJ2D - highJ2D);

    Paint paint = this.getProfilePaint();
    g2.setPaint(paint);
    g2.fill(body);

    g2.draw(body);

    EntityCollection entities = null;
    if (info != null) {
        entities = info.getOwner().getEntityCollection();
    }

    if(entities != null) {
        this.addEntity(entities, hotspot, dataset, series, item,  entityX: 0.0D,  entityY: 0.0D);
    }
}
```

Figure 3.4: drawRectangle() method

3.2 Demo Application

This section goes into detail regarding the implementation of demo application which MarketProfileRenderer.

3.2.1 How to create a MarketProfile chart with JFreeChart in three steps:

- create a dataset containing the data to be displayed in the chart;
- create a JFreeChart object that will be responsible for drawing the chart;
- draw the chart to some output target (a panel on the screen);

3.2.2 Dataset for Market Profile Chart

Utility class DataGenerator reads data from csv file and parses each row into a data object that contains date and time, open, high, low, close, volume values. Afterwards, all of the parsed data is added to be stored in some object of implementation of OHLCDataset interface.

3.2.3 JFreeChart object for MarketProfile Chart

A subclass of the abstract class Plot is the dependency to create an instance of JFreeChart class. For market profile charts, XYPlot is used. Dataset and renderer is injected into XYPlot.

```
private static JFreeChart createMPChart() throws Exception {
    // dataset
    OHLCDataset dataset = DataGenerator.generateDefaultHighLowDataset();

    // renderer
    MarketProfileRenderer renderer = new MarketProfileRenderer(DateTimeUtility.DEFAULT_DATE_TIME_FORMAT.parse( source: "2021-07-08 09:30:00"), drawVolume: true);

    // plot
    ValueAxis timeAxis = new DateAxis( label: "Time");
    NumberAxis valueAxis = new NumberAxis( label: "Price");
    valueAxis.setAutoRangeIncludesZero(false);
    valueAxis.setUpperMargin(0.00);
    valueAxis.setLowerMargin(0.00);
    XYPlot plot = new XYPlot(dataset, timeAxis, valueAxis, renderer);
    plot.setDomainPannable(true);

    return new JFreeChart( title: "MP Demo", JFreeChart.DEFAULT_TITLE_FONT, plot, createLegend: true);
}
```

Figure 3.5: creating JFreeChart instance

3.2.4 Drawing the chart on a panel on the screen

The final step is to display the chart somewhere. JFreeChart is very flexible about where it draws charts, thanks to its use of the Graphics2D class. A subclass of org.jfree.ui.ApplicationFrame class can be used as a panel, because ApplicationFrame extends JFrame from jswing library and used primarily to draw a chart panel.

```
public class MarketProfileDemo extends ApplicationFrame {

    public MarketProfileDemo(String title) throws Exception {...}

    public static JPanel createDemoPanel() throws Exception {...}

    private static JFreeChart createMPChart() throws Exception {...}

    public static void main(String[] args) throws Exception {
        MarketProfileDemo demo = new MarketProfileDemo( title: "demo");
        demo.pack();
        UIUtils.centerFrameOnScreen(demo);
        demo.setVisible(true);
        demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Figure 3.6: Overview of the demo source code

Chapter 4

Testing

In this section, the unit testing and manual user testing of the MarketProfileRenderer are discussed.

4.1 Unit tests

Unit-testing is integral part in developing a software solution that helps to discover misbehaviour of a software solution. However, MarketProfileRenderer is partially unit-tested where it was seen trivially. Since the main functionality of MarketProfileRenderer is to draw items on a chart with the help of Graphics2G, it was rather unclear how to test many parts of the implementation with unit-tests, therefore only limited number of methods were tested with unit-tests.

```
@Test
public void roundValueWithTickSizeTest() throws Exception {
    MarketProfileRenderer mock = PowerMockito.spy(new MarketProfileRenderer(dateMock, drawVolume: false));
    mock.setTickSize(0.05);
    BigDecimal exp = new BigDecimal( val: "0.15");
    BigDecimal res = Whitebox.invokeMethod(mock, methodToExecute: "roundValueWithTickSize", ...arguments: 0.16);
    Assert.assertEquals(exp, res);
}
```

Figure 4.1: Unit-test of the rounding method

4.2 User testing

Since the unit-testing of the main functionality was not achieved, the significant part of testing was focused on user manual testing. For the purpose of convenience both in utilizing the software solution and testing it, label generator provided by JFreeChart was added to the renderer. Label generator allows to display actual values of a data item to be displayed when hovering

```java
@Test
public void getSymbolTest() throws Exception {
    MarketProfileRenderer mock = PowerMockito.spy(new MarketProfileRenderer(dateMock, drawVolume: false));

    char exp = 'B';
    char res = Whitebox.invokeMethod(mock, methodToExecute: "getSymbol", ...arguments: 'A');
    Assert.assertEquals(exp, res);

    exp = 'Z';
    res = Whitebox.invokeMethod(mock, methodToExecute: "getSymbol", ...arguments: 'Y');
    Assert.assertEquals(exp, res);

    exp = 'a';
    res = Whitebox.invokeMethod(mock, methodToExecute: "getSymbol", ...arguments: 'Z');
    Assert.assertEquals(exp, res);

    exp = 'A';
    res = Whitebox.invokeMethod(mock, methodToExecute: "getSymbol", ...arguments: 'z');
    Assert.assertEquals(exp, res);
}
```

Figure 4.2: Unit-test of the symbol method

over the chart item. The contents of label are date and time, open, high, low, close and volume values of the data item.
For the simplicity of manual testing, small feeds of data were given to the renderer to display on the chart, which were also manually calculated and compared to values on the chart.

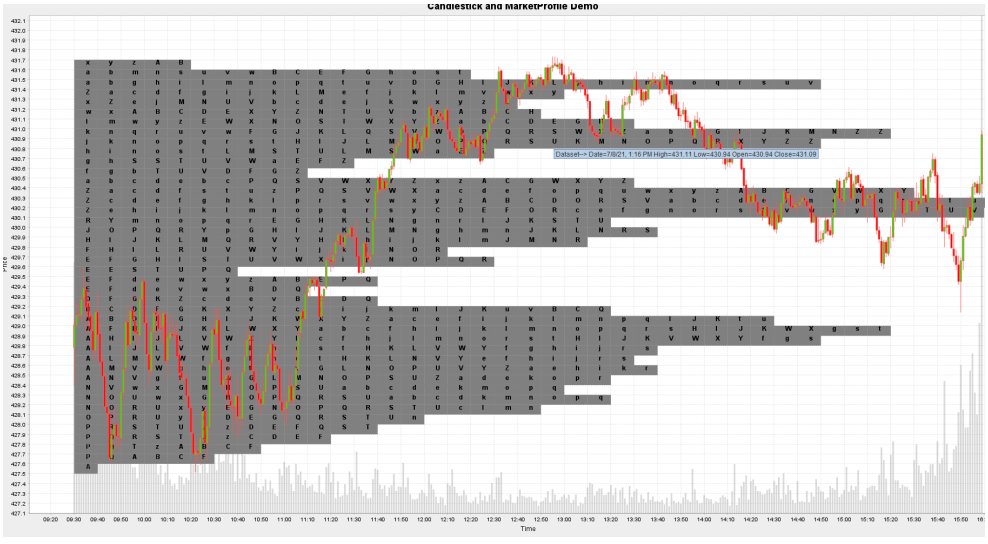

Figure 4.3: Example of Market Profile Chart with MarketProfileRenderer JFreeChart

4.3 Comparing MarketProfileRenderer with existing libraries/frameworks

People who use market profile charts become evangelists to the cause. They believe it offers unique insights into buy and sell opportunities. It can be a good option if you are a quick-fire day trader. On the downside, it can require serious effort to learn how to use them, and very few stock chart services offer this type of charting. Metastock, Optuma, ProRealTime, Overcharts and small number of other software applications provide functionality of drawing market profile charts.

4.3.1 MetaStock

MetaStock is used for charting and technical analysis of stock (and other asset) prices. It has both real-time and end-of-day versions. It allows setting TPO(time price opportunity) Color Mode, choosing start period, setting price interval, number of profiles to be drawn, shows label over TPO, which shows details on the item.

4.3.2 Overcharts

Overcharts is my favourite application that allows rendering market profile charts. It provides options to build chart on session(Trading Hours) defined in the reference chart or indicator settings and build "The Long-Term TPO Profile" represented by TPO Profiles built on the last x sessions, days, weeks, months or years. Additionally, it calculates and displays value area and point of control per profile.

4.3.3 MarketProfileRenderer with JFreeChart

The solution described in this thesis work is centered on the rendering market profiles with JFreeChart library. Another focus was to follow the structure of JFreeChart library and provide a renderer in a similar scope to other implementations of XYItemRenderer, like CandlestickRenderer. MarketProfileRenderer draws only one profile per dataset. To draw multiple profiles on the same chart separate datasets need to be provided and create a new instance of MarketProfileRenderer per dataset.
Another feature that MarketProfileRenderer lacks comparing to well-established software solutions similar to Overcharts that allow drawing market profile charts, is calculation and display of both value area and point of control.

Chapter 5

Conclusion

In this thesis, I analysed market profile charts and JFreeChart library. The analysis of how to read market profile charts and what features JFreeChart library already offers and how it renders charts allowed me to make a list of requirements that had to be implemented to accomplish the goals assigned to the extension of the JFreeChart with market profile. Moreover, implementations of XYItemRenderer interface allowed me to get a detailed understanding how JFreeChart renders items on an xy chart and provided me initial steps on designing and implementing MarketProfileRenderer.
I discussed how the data will be calculated to transform it from date and time to x value on domain axis and prices to y values on range axis and implemented the solution to MarketProfileRenderer. By creating a demo application, I have verified functionalities MarketProfileRenderer provides. Finally, I have compared my solution with existing solutions to discover difference of features provided. The final version of the market profile renderer and demo application provided by this work satisfy goals of the thesis on a high-level, although there are functionalities can be added as previously discussed.

5.1 Future Work

While provided solution renders market profile charts from open high low close and volume datasets and follows the structure of the JFreeChart library, there are functionalities could be added such as calculating and highlighting of value area and point of control. Another feature that would improve MarketProfileRenderer is to allow to render multiple profiles per chart, rather than initialising new instance for each profile.

Bibliography

1. *Market profile*. Wikimedia Foundation, 2020. Available also from: https://en.wikipedia.org/wiki/Market_profile.
2. *Market profile trading: The Most Comprehensive Guide*. 2021. Available also from: https://www.tradingriot.com/market-profile/#Before_we_begin.
3. TRADINGRIOT.COM. *marketprofile*. 2021. Available also from: https://tradingriot.com/wp-content/uploads/2020/09/mp1-e1603624598603.jpg.
4. *JFreeChart Home Page [online]*. 2005-2021. Available also from: https://www.jfree.org/jfreechart/. [Cited 2021-11-21].
5. *JFreeChart Downloads Page [online]*. 2005-2021. Available also from: https://www.jfree.org/jfreechart/download/. [See: jfreechart-1.0.0-install.pdf, Cited 2021-11-21].
6. *Candlestick Chart*. Wikimedia Foundation, 2022. Available also from: https://en.wikipedia.org/wiki/Candlestick_chart.
7. *Open-High-Low-Close Chart (OHLC Chart) - Learn about here*. [N.d.]. Available also from: https://datavizcatalogue.com/methods/OHLC_chart.html.
8. *Java (programming language)*. Wikimedia Foundation, [n.d.]. Available also from: https://en.wikipedia.org/wiki/Java_(programming_language).
9. *Java development kit*. Wikimedia Foundation, [n.d.]. Available also from: https://en.wikipedia.org/wiki/Java_Development_Kit.
10. *Opencsv users guide*. [N.d.]. Available also from: http://opencsv.sourceforge.net/.
11. *Comma-separated values*. Wikimedia Foundation, [n.d.]. Available also from: https://en.wikipedia.org/wiki/Comma-separated_values.

12. *Junit*. Wikimedia Foundation, [n.d.]. Available also from: `https://en.wikipedia.org/wiki/JUnit`.

13. *Powermockito*. [N.d.]. Available also from: `https://powermock.github.io/`.

Appendix A

Acronyms

GUI Graphical user interface

XML Extensible markup language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format