WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor's Thesis

Title of Bachelor's Thesis (English)	JBusinessChart Providing Charts to ooRexx
Title of Bachelor's Thesis (German)	JBusinessChart Bereitstellung von Diagrammen für ooRexx
Author (last name, first name):	Voglmüller, Jan
Student ID number:	12209774
Degree program:	Bachelor of Science (WU), BSc (WU)
Examiner (degree, first name, last name):	ao.Univ.Prof. Dr. Rony G. Flatscher

I hereby declare that:

1. I have written this Bachelor's thesis myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. This Bachelor's Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This Bachelor's Thesis is identical with the thesis assessed by the examiner.
4. (Only applicable if the thesis was written by more than one author): this Bachelor's thesis was written together with

   The individual contributions of each writer as well as the co-written passages have been indicated.

25.08.2024
Date

Jan Voglmüller
Signature

WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

JBusinessChart
Providing Charts to ooRexx

Jan Voglmüller

h12209774

Department of Information Systems & Society

Reviewer: ao.Univ.Prof. Dr. Rony G. Flatscher

25 August 2024

Table of Contents

List of Figures

List of Tables

List of Listings

Abstract

This thesis leverages the programming language ooRexx (Open Object Rexx) and its powerful framework BSF4ooRexx to develop and refine the creation of business charts. By integrating BSF4ooRexx, Java is made available, enabling, among other things, the use of Java2D for graphic rendering. To facilitate the generation of images in Java2D, the command handler JDOR (Java2D Drawing for ooRexx) eliminates the need to directly interact with Java classes.

To further enhance the usability of Java2D and to provide a more user-friendly approach to creating business charts specifically, this thesis introduces and demonstrates "JBusinessChart", a framework for ooRexx built on top of BSF4ooRexx and JDOR. It offers classes for creating various types of business charts, such as column, bar and line charts, with the option to combine and stack them. All the functionalities are illustrated through "nutshell examples".

1. Introduction

Open Object Rexx (ooRexx) is an open-source programming language managed by the Rexx Language Association (RexxLA) [1]. Initially, the Rexx language was developed by IBM as a scripting language [2]. Later, IBM introduced Object Rexx, which added object-oriented capabilities. Eventually, RexxLA took over the project, making it open source and maintaining it as the ooRexx we know today [2].

BSF4ooRexx850 (Bean Scripting Framework for Open Object Rexx) is a package that creates a bridge between ooRexx and Java, allowing users to easily implement Java methods in ooRexx without the need for adaptation, thanks to its seamless integration [2]. This hidden implementation effectively camouflages Java, making the framework user-friendly [3].

JDOR (Java2D Drawing for ooRexx), included in the BSF4ooRexx850 package, is a command handler that enables ooRexx programmers to create graphics using the powerful Java2D by using a new command language that is simple and understandable [4]. This allows them to leverage the full functionality of these Java classes in their ooRexx applications [2].

The aim of this bachelor thesis is 1) to develop JBusinessChart, a framework that builds upon JDOR to facilitates the creation of business charts by shielding ooRexx programmers from JDOR and 2) to explain and demonstrate the new capabilities that JBusinessChart adds. This framework aims to enhance the visual representation of data within ooRexx applications, making it easier for users to create professional and effective business charts.

2. Requirements

This chapter is about the requirements for this project. To write the code and run the nutshell examples, several programs are necessary. This section will explain the required programs and their installation process.

The following guide is tailored for Windows, but similar steps apply for other operating systems, with detailed instructions available through the provided links. While the installation order is not strictly mandatory, following the recommended sequence is advised to avoid potential issues that may arise from installing the programs in a different order. If errors occur, reinstalling the programs might resolve them.

Chapter 2.4. is an alternative to chapters 2.2. and 2.3. and offers a less complicated way to run the nutshell examples. However, for the development process the permanent programming environment is recommended.

2.1. Java

Java 8 or higher is required for BSF4ooRexx850 to function properly. To utilize the full functionality of the framework, a Java version with JavaFX support is needed. The following link offers such a Java version from BellSoft:

https://bell-sw.com/pages/downloads/#jdk-22 [5]

After accessing this link, the "Windows" section can be found by scrolling down. Here, the Java version with JavaFX support can be selected by changing the package option to "Full JDK", as shown in Figure 1. Clicking on "MSI" will download the installer.

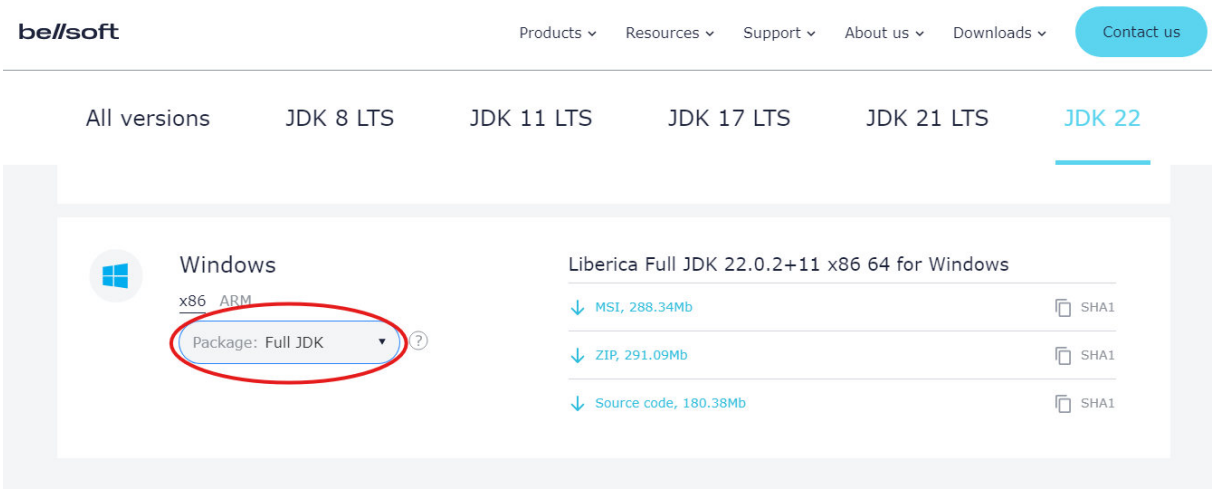

Figure 1: Download Java

After downloading the MSI file, it must be executed to start the setup process. Various options, such as changing the storage location, are available, but all settings should remain at their default values. To complete the installation, navigate through the menus as illustrated in Figure 2.

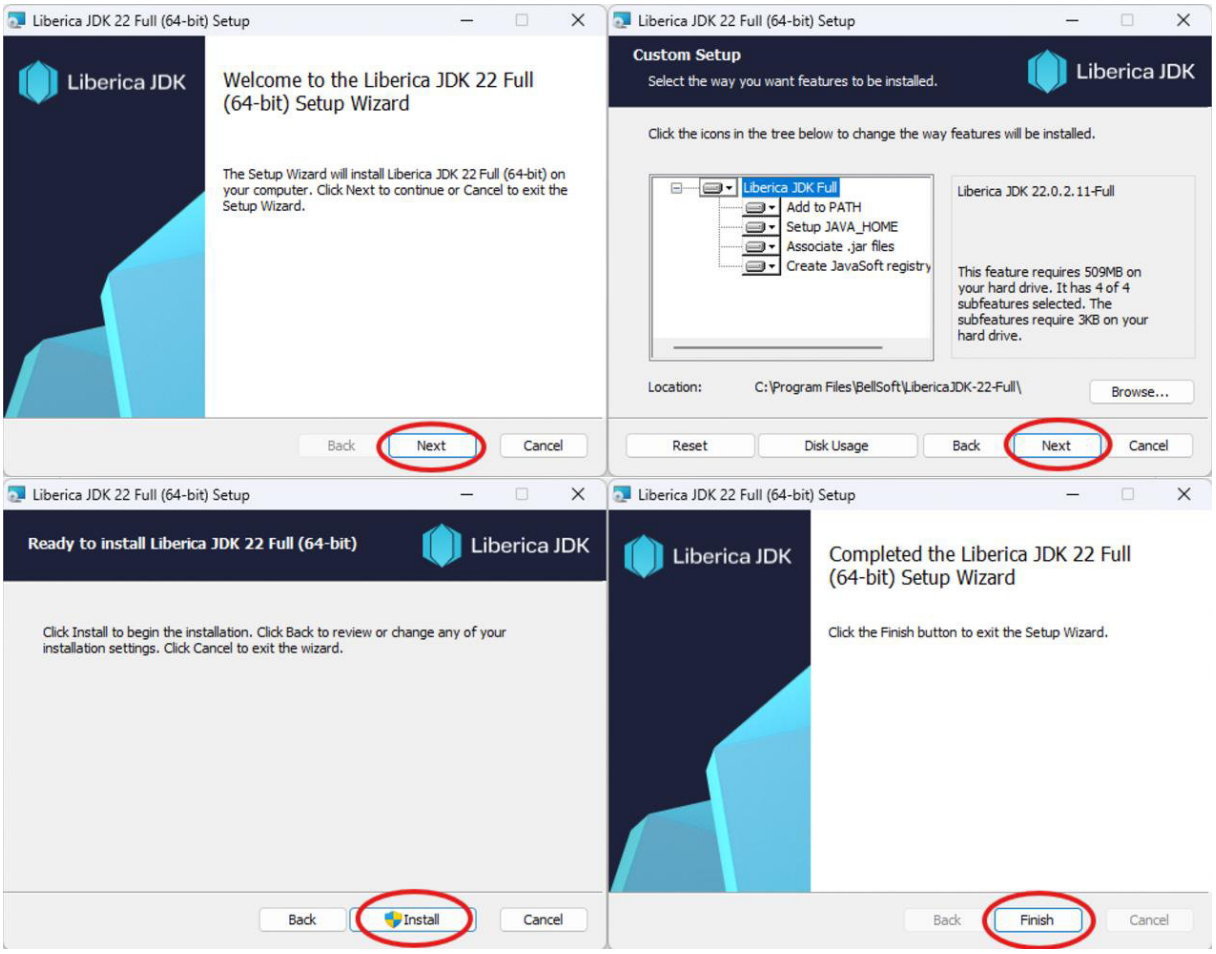

Figure 2: Java Installation

To verify the correct installation of Java, a command window can be used to query the installed version using the command "java -version". The displayed version should match the version that was just downloaded and installed. An example can be seen in Listing 1.

```
openjdk version "22.0.2" 2024-07-16
OpenJDK Runtime Environment (build 22.0.2+11)
OpenJDK 64-Bit Server VM (build 22.0.2+11, mixed mode, sharing)
```

Listing 1: Terminal Command "java -version"

2.2. Open Object Rexx 5.1.0

Now, the programming language itself needs to be installed. Version 5.0.0 or later of ooRexx, with a 64-bit architecture, is required. The most recent version, 5.1.0, is recommended. This version can be accessed via the following link for Windows:

https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/oorexx-5.1.0-12859.windows.x86_64.exe/download [6]

The file for other operating systems can be found on the following website by selecting and clicking the corresponding link:

https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/ [7]

The provided link [6] for Windows will automatically start the download after 5 seconds, or the download can be initiated manually by clicking "Download." Before executing the downloaded file, it must be unblocked or de-quarantined to ensure a smooth installation process. Otherwise, the installation might be blocked by the operating system because the source is not verified by Windows. To unblock the file, right-click on the .exe file and select "Properties" (German: "Eigenschaften"). In the new window, go to the "General" tab (German: "Allgemein") and locate the "Unblock" checkbox (German: "Zulassen") at the bottom right, refer to Figure 3. Check this box and confirm by clicking "OK".

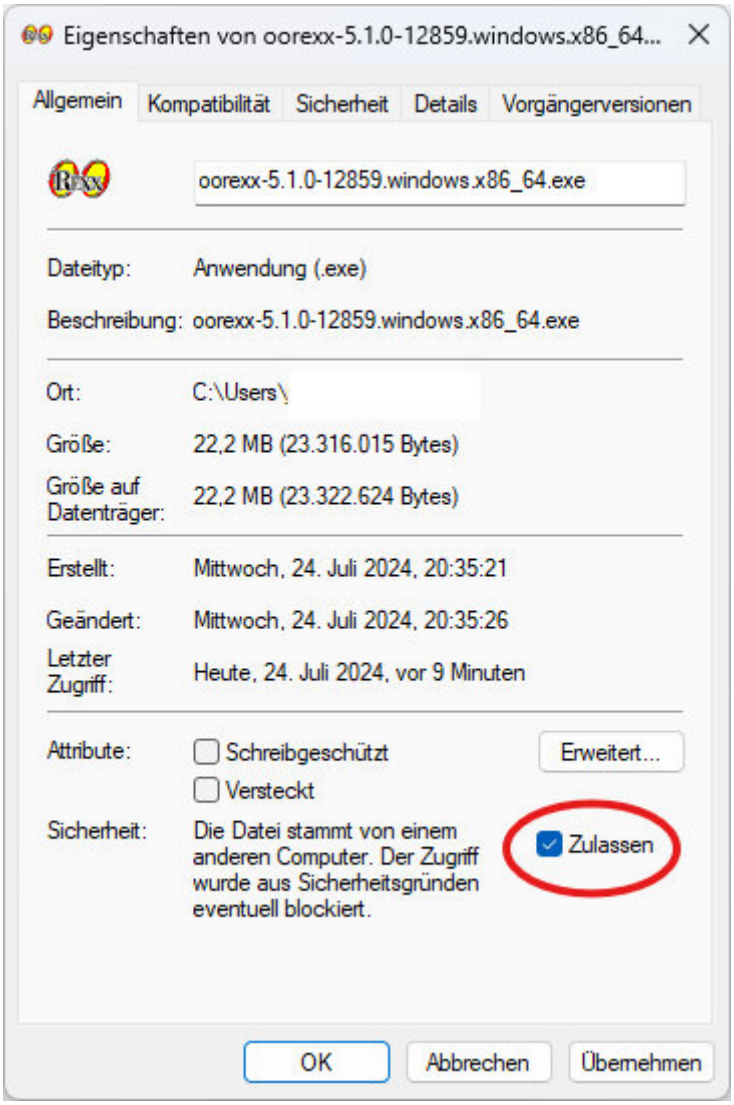

Figure 3: De-quarantine ooRexx

Another option is to open a command line window and run "powershell Unblock-File <filename>", replacing <filename> with the actual name of the downloaded file, in the folder where the file is located.

For macOS operating systems, the de-quarantining must be done in a terminal window. Use the command "xattr -d com.apple.quarantine <filename>", replacing <filename> with the actual name of the downloaded file, to accomplish this.

The next step is to start the setup by executing the downloaded file. As with the previous installation, all preferences should remain unchanged, allowing the user to proceed with the default settings. If an earlier version of ooRexx is already installed, the installer will offer an option to uninstall the older version; this option should be selected and all subsequent preferences should also remain at their default values. After clicking through the menus, the installation will be completed.

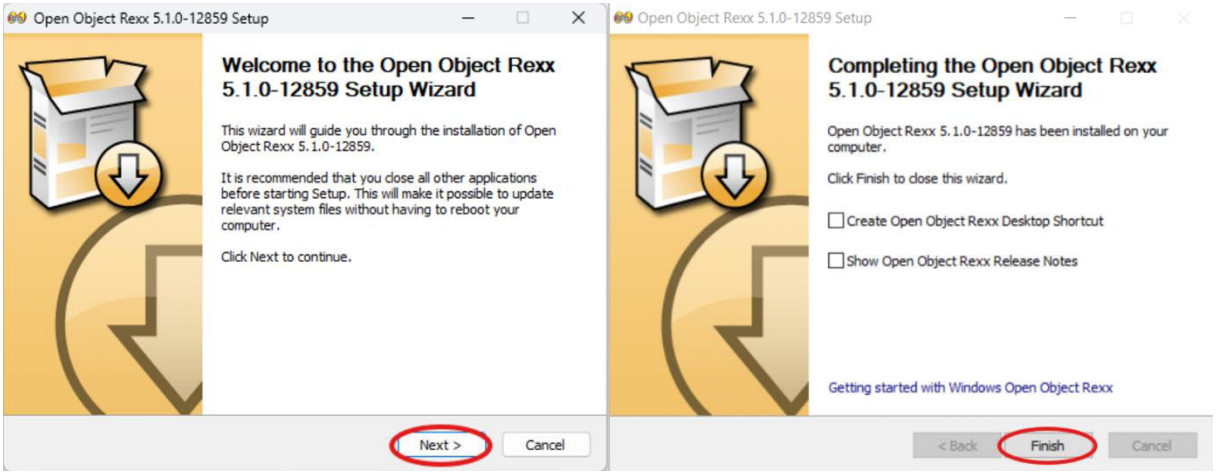

Figure 4: ooRexx Installation

Once the installation is complete, the correctness of the installation and the version can be verified again in the command window, this time using the command "rexx -version". The result should look like Listing 2.

```
Open Object Rexx Version 5.1.0 r12859
Build date: Jul 21 2024
Addressing mode: 64
Copyright (c) 1995, 2004 IBM Corporation. All rights reserved.
Copyright (c) 2005-2024 Rexx Language Association. All rights reserved.
This program and the accompanying materials are made available under the terms
of the Common Public License v1.0 which accompanies this distribution or at
https://www.oorexx.org/license.html
```

Listing 2: Terminal Command "rexx -version"

2.3. BSF4ooRexx850

The final step is to install the bridge between ooRexx and Java. A download link is provided, which will automatically start the download after 5 seconds.

https://sourceforge.net/projects/bsf4oorexx/files/latest/download [8]

As with the programming language installation, the file must be unblocked before execution to ensure a smooth installation process. Follow the same unblocking procedure as previously described and confirm by clicking "OK." For reference, see Figure 5.

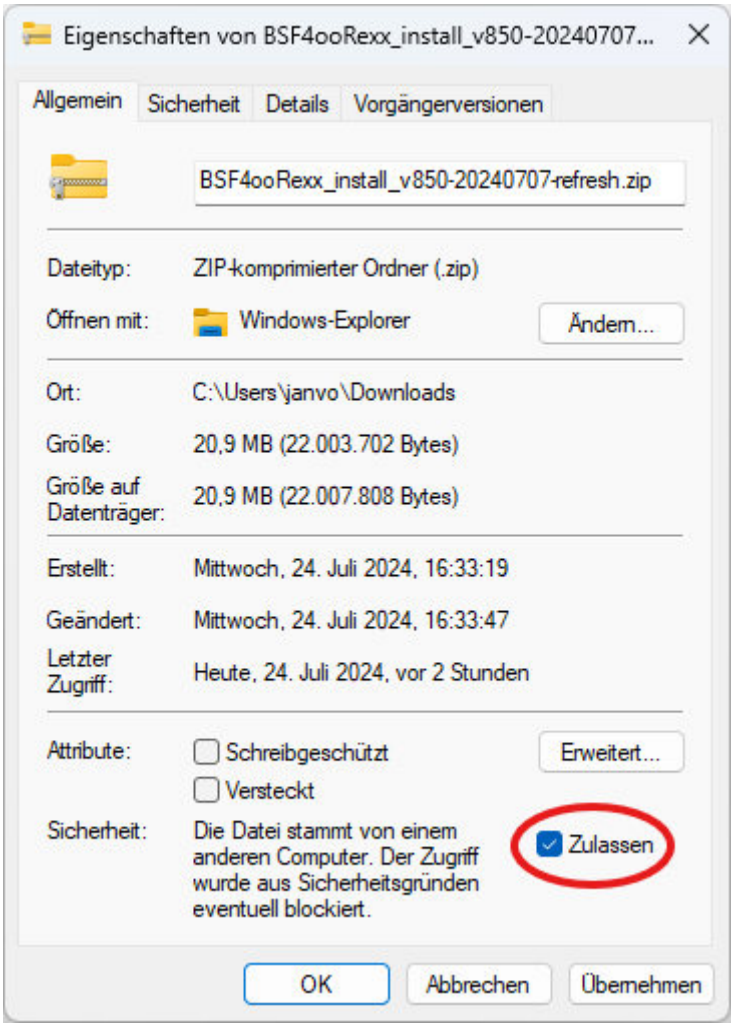

Figure 5: De-quarantine BSF4ooRexx

For Windows, the command line option is available once again. The command “powershell Unblock-File <filename>” remains unchanged.

After unblocking the file, the ZIP must be unzipped into a new folder. Inside this new folder, there is a subfolder named "install," which contains separate folders for each operating system. For this project, execute the “install.cmd” file located within the "windows" folder to start the installation process. A command window will appear.

After completing all steps and ensuring no errors occurred, as shown in Figure 6, the installation is finished.

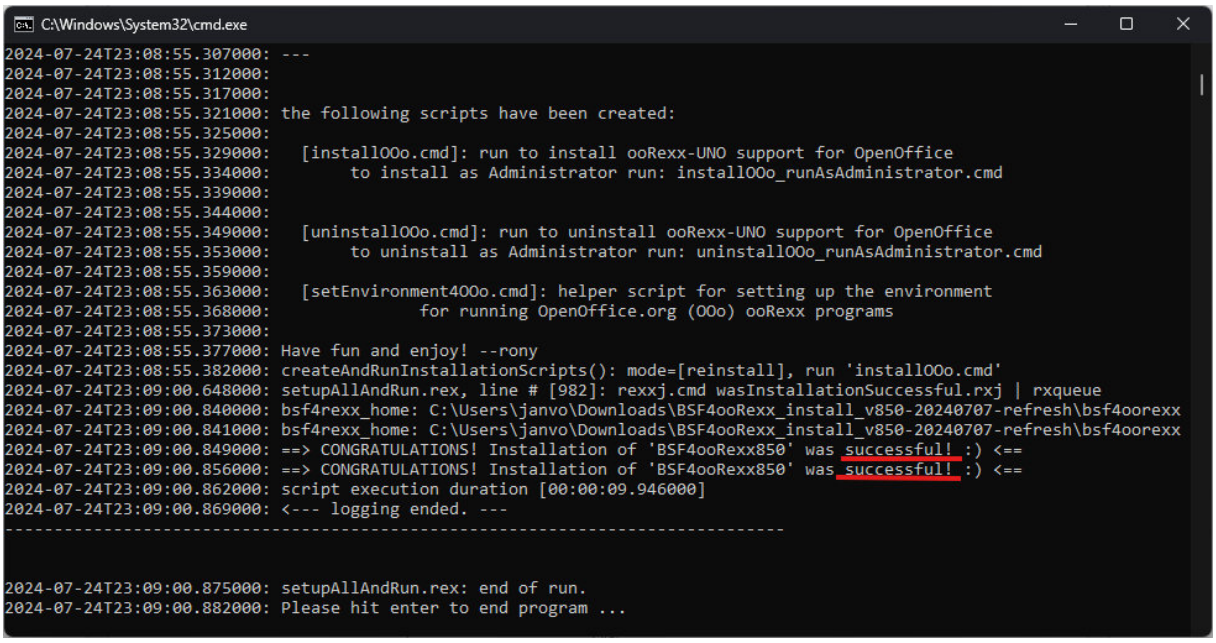

Figure 6: BSF4ooRexx Installation

Once again there is the possibility to verify that everything is working correctly. If executing "ooRexxTry.rxj" in the folder "C:\Program Files\BSF4ooRexx850\utilities" opens the graphical user interface as shown in Figure 7, then all requirements have been met.

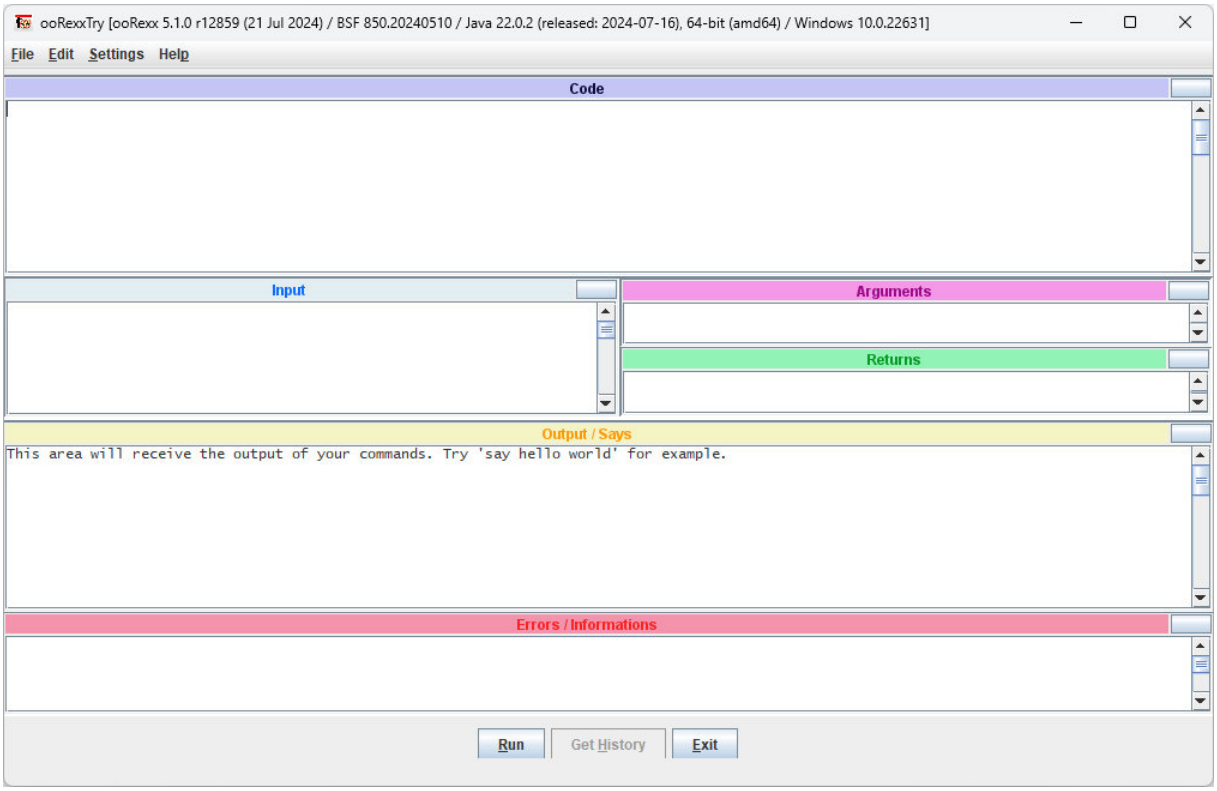

Figure 7: Output ooRexxTry.rxj

2.4. Portable ooRexx (Alternative)

Instead of installing the runtime environment for the programming language and its framework "BSF4ooRexx" permanently, a temporary setup for ooRexx commands can be created through a portable version. This allows skipping the instructions described in chapters 2.2. and 2.3. and provides a simple way to execute the ooRexx nutshell examples. The following link provides the latest version for Windows in form of a ZIP file:

https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/portable/oorexx-5.1.0-12860.windows.x86_64-portable-release.zip/download [9]

Files for other operating systems can be found under:

https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/portable/ [10]

After the download is completed, the file needs to be unblocked. Right-click on the ZIP archive and select "Properties" (German: "Eigenschaften"). In the new window, under "General" (German: "Allgemein"), the "Unblock" checkbox (German: "Zulassen") can be found at the bottom right. By checking and confirming this option with "OK", the file gets de-quarantined. Refer to Figure 8.

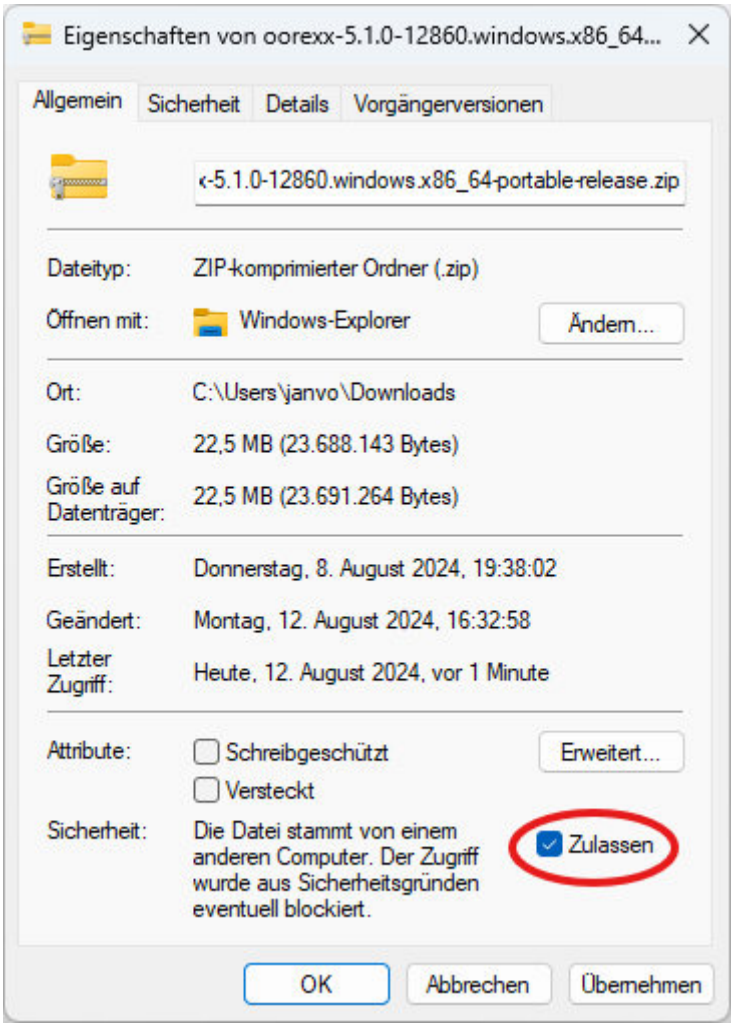

Figure 8: De-quarantine ooRexx-Portable

Another option is to open a command line window and run "powershell Unblock-File <filename>", replacing <filename> with the actual name of the downloaded file, in the folder where the file is located.

For macOS operating systems, the de-quarantining must be done in a terminal window. Use the command "xattr -d com.apple.quarantine <filename>", replacing <filename> with the actual name of the downloaded file, to accomplish this.

After unblocking the ZIP file and extracting its contents, the installation process can be started by executing "setupoorexx.cmd" for Windows. Two new CMD files should appear: "rxenv.cmd" and "setenv2rxenv.cmd".

The first file provides the user the possibility to execute ooRexx programs. To do so, a new command line window needs to be opened in the folder with the mentioned "rxenv.cmd" file. Simply right-click and select "Open in Terminal" in the folder or use the command "cd" followed by the folder path to change the directory. In the command line window, programs

can be executed by sending ".\rxenv rexx" followed by the program's name, for testing purpose "testoorexx.rex". The executed program needs to be in the mentioned folder.

3. Architecture

JBusinessChart is designed to provide a straightforward business chart creation process with minimal steps. The goal during the development was to create a simple logic for the framework, that is as intuitive as possible: The programmer can create simple charts and later modify or expand them. This method enables fast results while also avoiding restrictions on the user. It facilitates the creation of more complex business charts by combining, mixing or stacking these simple charts.

The UML (Unified Modeling Language) Class diagram in Figure 9 provides an overview of JBusinessChart, including its classes, interconnections, attributes and methods. In the following subsections, the classes will be explained in more detail and tables with the most important attributes and methods for the user will be provided.

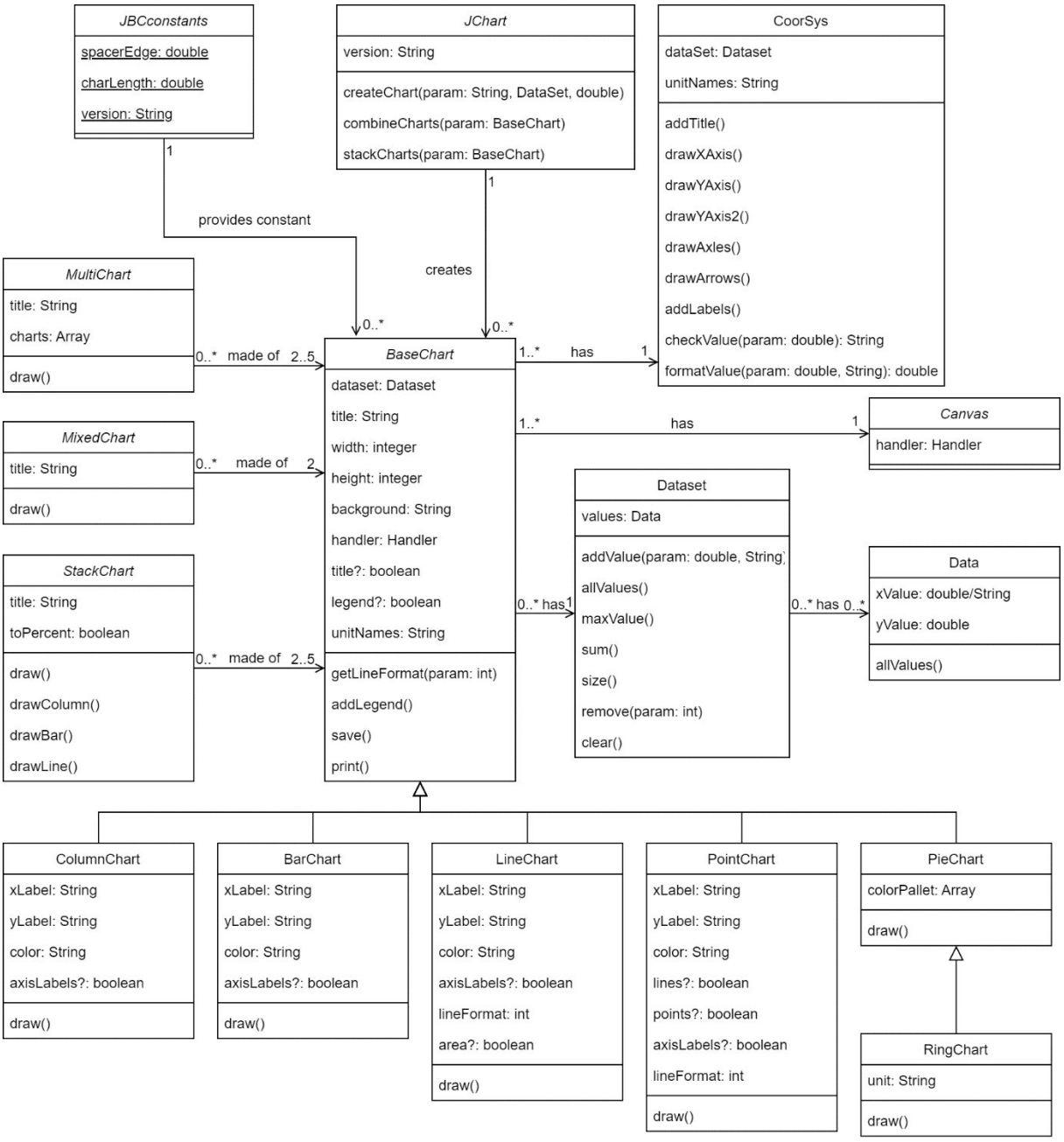

Figure 9: UML Class Diagram

3.1. Data

The "Data" class represents a two-dimensional data point. Each data point contains an "x" value and a "y" value, which can be accessed by the corresponding attributes. While "y" values must be positive numbers, "x" values can also be strings for specific chart types. For example, in a column chart, the x-axis is used for naming the displayed values, so only the y value needs to be numeric. In a point chart, however, the points are placed in a two-dimensional coordinate system based on x and y, requiring both values to be numeric. JBusinessChart, in its first version and designed primarily for business charts, does not support negative inputs. For most business-related applications, positive numbers are

sufficient, which is why the input is currently limited to positive numeric values. The instances and methods of the class are utilized by the "DataSet" class and will therefore never be used directly by the user. To avoid overwhelming the user, this class is hidden.

3.2. DataSet

The "DataSet" class, once instantiated, is an array of instances of the "Data" class. The "DataSet" class is one of two public classes. It manages the use of the "Data" class by allowing the programmer to add instances of the "Data" class with the method "addValue" and saves them in its own array under the attribute "values". If a negative value is entered, it will be transformed to a positive number, to comply with the "Data" class. The "DataSet" class provides additional methods for managing the dataset. The method "allValues" displays all current data points in the dataset. With the "remove" method, a specific two-dimensional data point, specified by its position, can be removed. All other values will move up. With "clear", all values can be removed. All further methods are for internal use and will not generate any say instruction output.

This structure allows any chart type to use the same dataset, enabling the user to display the same data in different formats. The "addValue" method can be used in combination with a loop to automatically add data from various sources, such as a JSON file.

Method	**Arguments**	**Description**
addValue	Double *xValue yValue*	Adds a two-dimensional data point, specified by its x and y values, to the "DataSet".
allValues		Displays all current values of the dataset.
maxValue	String *XorY*	Returns the maximum x or y value of the dataset, according to the provided argument.
sum		Returns the sum of all y values in the dataset.
size		Returns the size of the dataset.
remove	Integer *positionNumber*	Removes the value at the specified position and all subsequent values adjust accordingly.
clear		Wipes all data from the dataset.

Table 1: Methods DataSet

3.3. JChart

The “JChart” class serves as the primary interface for the programmer. It accepts nearly all commands for creating charts and distributes them to the appropriate classes. The “JChart” itself primarily functions as a dispatcher, handling the routing of commands and simplifying the process for the user by selecting the correct methods and classes, therefore, it is made public. The actual work of creating business charts is done by other classes mentioned later in this paper.

Although the “JChart” is so important, only three methods are available, which further simplifies the handling of JBusinessChart. The “createChart” method is used by the programmer to create any type of simple chart. The first argument is the chart type entered as a string in Pascal Case [11]. Possible values are: “ColumnChart”, “BarChart”, “LineChart”, “PointChart”, “PieChart” and “RingChart”. This is followed by the nickname for a previously created dataset. All subsequent arguments are optional and can be modified later. The third argument, if provided, is a String with the title of the chart, followed by Strings for the names of the x and y axes. The last two arguments are numbers that define the width and height of the chart when displayed.

The “combineCharts” method provides two functionalities. At least two arguments in form of previously created charts need to be provided, but up to five are possible. If two or more charts of the same chart type are entered, “JChart” will merge them into a single chart and create a “MultiChart” (refer to 3.13.). If the user provides a column chart and a line chart, the order is irrelevant, the second functionality will be executed automatically, creating a “MixedChart” (refer to 3.14.) by overlaying the column and line charts and adding an additional y-axis to accurately represent both datasets. This dual functionality reduces complexity and enhances usability once again.

The last method, called “stackCharts”, will accept two to five charts of the same type. This method will stack the different charts on top on each other to create a “StackChart” (refer to 3.15.).

Method	Arguments	Description
createChart	String *chartType* *"ColumnChart", "BarChart", "LineChart", "PointChart", "PieChart", "RingChart"* DataSet *nickname* [optional] String *title xLabel yLabel* Integer *width height*	Creates a chart object of the specified chart type with the provided dataset. If the optional arguments are not provided, they are set to default values.
combineCharts	Chart *nickname1 name2* [optional] Chart *name3 name4 name5*	Creates a new "MultiChart" object by overlaying the provided charts.
	ColumnChart nickname LineChart nickname	Creates a new "MixedChart" object from the provided column and line charts.
stackCharts	Chart *nickname1 name2* [optional] Chart *name3 name4 name5*	Creates a new "StackedChart" object by stacking the provided charts on top of each other.

Table 2: Methods JChart

3.4. BaseChart

For the user, the "JChart" class is the primary interface, however, the real center of this framework is the "BaseChart" class. It is not accidentally located in the middle of Figure 9, even though the user will never directly use an instance of this class. It is required for the internal architecture and is hidden from the user. It possesses all the fundamental attributes and methods required for a business chart.

The attributes "dataset", "title", "width" and "height" were already mentioned. The attribute "background" is used to store and modify the background color of the chart. JDOR offers predefined colors like "white", "black" and "pink" (refer to the command section in [4]). However, as JBusinessChart is built on JDOR, all JDOR commands can also be used, including creating custom colors. The "handler" attribute stores the object of the JDOR handler used for the chart. This allows different internal functionalities and provides users with knowledge of JDOR additional possibilities beyond the functional area of JBusinessChart.

Attributes with a question mark at the end, such as "title?" and "legend?", store Boolean values and are used for the option to hide or display the corresponding element of a chart. The attribute "unitNames" holds the default name for large numbers as a String in an array and can be changed to suit a different language. The method "save" allows a user to save a

chart in the local home directory as a PNG and the "print" method will open the default printer window with the business chart ready to print. The other methods are again for internal use.

Attribute	Arguments	Description
dataset	DataSet *nickname*	Determines the data used for the chart.
title	String *title*	Determines the title of the chart.
width	Integer *width*	Determines the width of the window.
height	Integer *height*	Determines the height of the window.
background	String *color*	Determines the background color.
handler		Provides access to the chart's handler.
title?	Boolean *truthValue*	If set to true, the title will be displayed.
legend?	Boolean *truthValue*	If set to true, the legend will be displayed.
unitNames	Array *unitName*	Determines the names for large numbers based on their position in the array: [1] < 1,000; [2] < 1,000,000; [3] < 1,000,000,000; [4] < 1,000,000,000,000; [5] < 1,000,000,000,000,000.

Table 3: Attributes BaseChart

Method	Arguments	Description
save	[optional] String *saveLocation*	Saves the chart as a PNG file in the local home directory or at the specified location. The chart must be constructed beforehand using the draw method.
print		Opens the default printer window with the chart prepared for printing.

Table 4: Methods BaseChart

3.5. Canvas

The "Canvas" class has the purpose to create a window, including a drawing canvas, to later draw and display all required elements of a business chart. Because every chart needs a canvas, the process is always exactly the same and therefore this class can be used to reduce redundancy. The entire work is done during the instantiation, which is why there are no methods. The only attribute is used to store and forward the handler to later make it accessible through the chart object. This allows the "Canvas" class to be hidden from the user.

3.6. CoorSys

The "CoorSys" class is also used to reduce redundancy. It combines similar tasks related to preparing the canvas for the business chart. The class creates the coordinate system and other important elements. The methods divide the functionalities to increase modularity. The "addTitle" method adds a title to the chart, the "drawAxles" method will draw the corresponding axis and so on. The "checkValue" and "formatValue" methods are used to display large numbers correctly by reducing the zeros and adding a notation, such as indicating that the numbers are in millions.

All these functions are used by different chart classes allowing them to select which elements they need. Again, this class is only for internal usage and is not public. The attributes "Dataset" and "unitName" correspond to the attributes of the "BaseChart" class and serve for memory purposes.

3.7. ColumnChart

The "ColumnChart" class is used to represent and draw a column chart. It is instantiated and fed with all important data by "JChart". The "ColumnChart" class inherits all the attributes and methods from its superclass "BaseChart" and has additional attributes: "xLabel" and "yLabel" for describing the x and y axes, "color" to define and modify the column color and "axisLabels?" to determine if the labels on the axes should be visible.

The class is not public, but as mentioned, an instance can be created through "JChart" and saved under a nickname. With this nickname, the chart object can be accessed, constructed and displayed on the screen using the "draw" method.

Attribute	**Arguments**	**Description**
xLabel	String *xLabel*	Determines the description of the x-axis.
yLabel	String *yLabel*	Determines the description for the y-axis.
color	String *color*	Determines the color of the columns.
axisLabels?	Boolean *truthValue*	If set to true, the labels are displayed on the axes.

Table 5: Attributes ColumnChart

Method	**Arguments**	**Description**
draw		Constructs the chart and visualizes it on the screen.

Table 6: Methods ColumnChart

3.8. BarChart

The "BarChart" class works exactly the same as the "ColumnChart" class. The only slight difference is that it represents a bar chart instead of a column chart. Therefore, the color refers to the bars rather than the columns.

Attribute	Arguments	Description
xLabel	String *xLabel*	Determines the description of the x-axis.
yLabel	String *yLabel*	Determines the description for the y-axis.
color	String *color*	Determines the color of the bars.
axisLabels?	Boolean *truthValue*	If set to true, the labels are displayed on the axes.

Table 7: Attributes BarChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 8: Methods BarChart

3.9. LineChart

The "LineChart" class is similar to the "ColumnChart" and "BarChart" classes. It represents a line chart and builds on top of its superclass "BaseChart". It inherits the additional attributes from "ColumnChart" and "BarChart" and includes "lineFormat" to change the visual appearance of the line and "area?" to fill with the line color the area below the line. As with the other chart classes, the "draw" method is used to visualize the chart on the screen.

Attribute	Arguments	Description
xLabel	String *xLabel*	Determines the description of the x-axis.
yLabel	String *yLabel*	Determines the description for the y-axis.
color	String *color*	Determines the color of the lines and points.
axisLabels?	Boolean *truthValue*	If set to true, the labels are displayed on the axes.
lineFormat	Integer *formatNumber* 0: solid line, 1: dashed, 2: dotted, 3: mixed	Determines the appearance of the line.
area?	Boolean *truthValue*	If set to true, the area below the line will be filled.

Table 9: Attributes LineChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 10: Methods LineChart

3.10. PointChart

The "PointChart" class represents and draws a point chart. It draws points and, optionally, lines connecting them. It differs from other chart types because it uses the x and y value of data points to calculate their position in a two-dimensional coordinate system. In addition to the attributes and methods of its superclass, it possesses the attributes "lines?" to toggle the visibility of lines, "points?" to toggle the visibility of the points and the previously mentioned "xLabel", "yLabel", "color", "axisLabels?" and "lineFormat".

Attribute	Arguments	Description
xLabel	String *xLabel*	Determines the description of the x-axis.
yLabel	String *yLabel*	Determines the description for the y-axis.
color	String *color*	Determines the color of the lines and points.
lines?	Boolean *truthValue*	If set to true, the points will be connected by lines.
points?	Boolean *truthValue*	If set to true, the points will be displayed.
axisLabels?	Boolean *truthValue*	If set to true, the labels are displayed on the axes.
lineFormat	Integer *formatNumber* 0: solid line, 1: dashed, 2: dotted, 3: mixed	Determines the appearance of the line.

Table 11: Attributes PointChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 12: Methods PointChart

3.11. PieChart

The "PieChart" class represents and draws a pie chart. This class is a subclass of the "BaseChart" class and extends it with the attribute "colorPallet". This attribute is used to define the colors of the segments of the chart. The colors are stored in an array and are used in the corresponding order until the end is reached, at which point the cycle will repeat. By default, the colors are chosen randomly. The "draw" method once again serves to display the chart on the screen.

Attribute	Arguments	Description
colorPallet	Array *nickname*	Determines the colors of the segments in the pie chart in the specified order. If the array is too small, the colors will be reused.

Table 13: Attributes PieChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 14: Methods PieChart

3.12. RingChart

The "RingChart" class builds on top of its superclass "PieChart" and represents a ring chart. It adds additional elements to the pie chart, making it very similar. One important feature is that the total amount, visualized as a percentage in the chart, is displayed in the middle. The user can change the unit of this number by using the "unit" attribute and setting the desired String.

Attribute	Arguments	Description
unit	String *unitName*	Determines the unit of the total amount displayed in the centre of the ring chart.

Table 15: Attributes RingChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 16: Methods RingChart

3.13. MultiChart

As previously mentioned, the "MultiChart" class represents a single chart made of multiple charts of the same type laid over each other. It follows that the "MultiChart" class is responsible for drawing this multi chart. It remains hidden from the user as it is made available through "JChart". The "MultiChart" class utilizes the chart class it is built from to draw itself. It manages the sequence of the different charts and determines the elements needed by using the "draw" method along with an additional argument called "combiNum".

The instance of the "MultiChart" class will adopt the visual representation of the first provided chart. For example, if the primary chart hides the legend, the "MultiChart" object also hides it. The attributes "title" and "charts" are used to provide access to relevant information that might be subject to modification. The "charts" attribute holds all the charts that constitute the multi chart.

Attribute	Arguments	Description
title	String *title*	Determines the title of the chart.
charts	Array *nickname*	Contains all the charts the multi chart is made of.

Table 17: Attributes MultiChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 18: Methods MultiChart

3.14. MixedChart

The "MixedChart" class represents a chart made of both a column chart and a line chart. Similar to the "MultiChart" class, is utilizes the methods of the provided charts to display itself through the "draw" method. The "MixedChart" class manages the generation of both charts and adds additional information, such as a second y-axis, to correctly display all values.

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 19: Methods MixedChart

3.15. StackChart

The "StackChart" class represents and draws a single chart by stacking charts of the same type on top of each other. The programmer uses the "draw" method to display the chart on the screen. Internally, the "StackChart" instance selects the appropriate "draw" method, as there are several variations. Similar to the "MultiChart" class, it includes the "title" attribute to modifying the chart's title and the "toPercent" Boolean value to toggle the representation to percent.

Attribute	Arguments	Description
title	String *title*	Determines the title of the chart.
toPercent	Boolean *truthValue*	If set to true, the presented values will be displayed as percentages.

Table 20: Attributes StackChart

Method	Arguments	Description
draw		Constructs the chart and visualizes it on the screen.

Table 21: Methods StackChart

3.16.JBCconstants

The “JBCconstants” class provides the framework with three constants, ensuring consistency across all other classes. The first static attribute is “spacerEdge,” which is used for aesthetic purposes and determines the distance between the chart and the window's border. The second constant, “charLength,” represents the approximate width of a character displayed in the chart, essential for accurately positioning text. The final constant is a string that holds the current version of the framework.

4. Nutshell Examples

The nutshell examples serve to demonstrate the functionalities of JBusinessChart and help new users to understand the framework's logic. The examples are designed to ensure quick progress and produce initial results efficiently.

The examples are divided into multiple sections, each increasing in complexity to gradually introduce the user to JBusinessChart. Throughout all sections, nearly all possibilities and functionalities of the framework are showcased. To avoid overwhelming the user, visual changes are spread across the examples, keeping each example as concise as possible.

4.1. Create a Dataset

The first nutshell example demonstrates the use of the "DataSet" class, which forms the foundation for all other examples. In line 2 of Listing 3, a new dataset is created by sending the class the "new" message and assigning a nickname to the new object, in this case "myData". In lines 3 to 7, values are added using the "addValue" method, with the first argument representing the x value and the second representing the y value. With these steps, the dataset is now complete.

To display all current values, the "allValues" method can be used, as shown in line 11 of Listing 3. Listing 4 shows the output of the say instruction in the command line window. Additionally, the "size" and "maxValue" methods are demonstrated, however, these methods are typically used internally and are of less interest to the user.

Not only is adding values possible. To manage the dataset two options are offered. 1) Based on its position number a value can be removed from the dataset by using the "remove" method, demonstrated in line 21 of Listing 3. 2) All data can be wiped from the dataset by sending "clear" to the corresponding dataset.

For demonstration purposes, the current values are displayed after each operation and the system will always pause for a few seconds, as shown in lines 17 and 26. Line 39 of Listing 3 is particularly important because it includes the "requires" directive, which enables the code to access JBusinessCharts and all its classes.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
```

```
 8
 9  /* diplay all values of a dataset */
10  say "Displaying current values of myData..."
11  myData~allValues
12  myData~size
13  myData~maxValue
14
15  /* waiting 4 seconds */
16  say "waiting 4 seconds"
17  call syssleep 4
18
19  /* editing data */
20  say "Removing the third data of myData..."
21  myData~remove(3)
22  myData~allValues
23
24  /* waiting 4 seconds */
25  say "waiting 4 seconds"
26  call syssleep 4
27
28  /* deleting all data */
29  say "Deleting all data of myData and adding one..."
30  myData~clear
31  myData~addValue("2022", 1004)
32  myData~allValues
33
34  /* close windows after 20 seconds */
35  say "waiting 20 seconds"
36  call syssleep 20
37
38  /* get access to JBusinessChart classes */
39  ::requires 'JBusinessChart.cls'
```

Listing 3: 00_create_dataset.rxj

```
Displaying current values of myData...
#1 Values: 2012 10
#2 Values: 2013 15
#3 Values: 2014 25
#4 Values: 2015 40
#5 Values: >2015 45
waiting 4 seconds
Removing the third data of myData...
#1 Values: 2012 10
#2 Values: 2013 15
#3 Values: 2015 40
#4 Values: >2015 45
waiting 4 seconds
Deleting all data of myData and adding one...
#1 Values: 2022 1004
waiting 20 seconds
```

Listing 4: Output 00_create_dataset.rxj

4.2. Column Chart

This nutshell examples demonstrates the creation of a column chart. In the first seven lines of Listing 5, a dataset is created, as described in section 4.1. Line 10 shows the use of the "createChart" method of the "JChart" class to create a column chart object. The two required arguments "chartType" and "dataset" are provided, along with the first three optional arguments: "title", "xLabel" and "yLabel". Line 11 illustrates how to use this method. Line 12 demonstrates a customisation option by changing the "color" attribute from the default

value “lightblue” to “lightgreen”. By sending “draw” to the chart object, the business chart is visualised on the screen, as shown in Figure 10.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
8
9  /* creating classic column chart & display it */
10 myColumnChart = .JChart~createChart("ColumnChart", myData, "VBS Floridsdorf", "year", "students")
11                                          --chartType, dataSet [, title, xLabel, yLabel, width, height]
12 myColumnChart~color = lightgreen       --change bar color to "lightgreen" (default: "lightblue")
13 myColumnChart~draw
14
15 /* close windows after 20 seconds */
16 say "waiting 20 seconds"
17 call syssleep 20
18
19 /* get access to JBusinessChart classes */
20 ::requires 'JBusinessChart.cls'
```

Listing 5: 01_column_chart.rxj

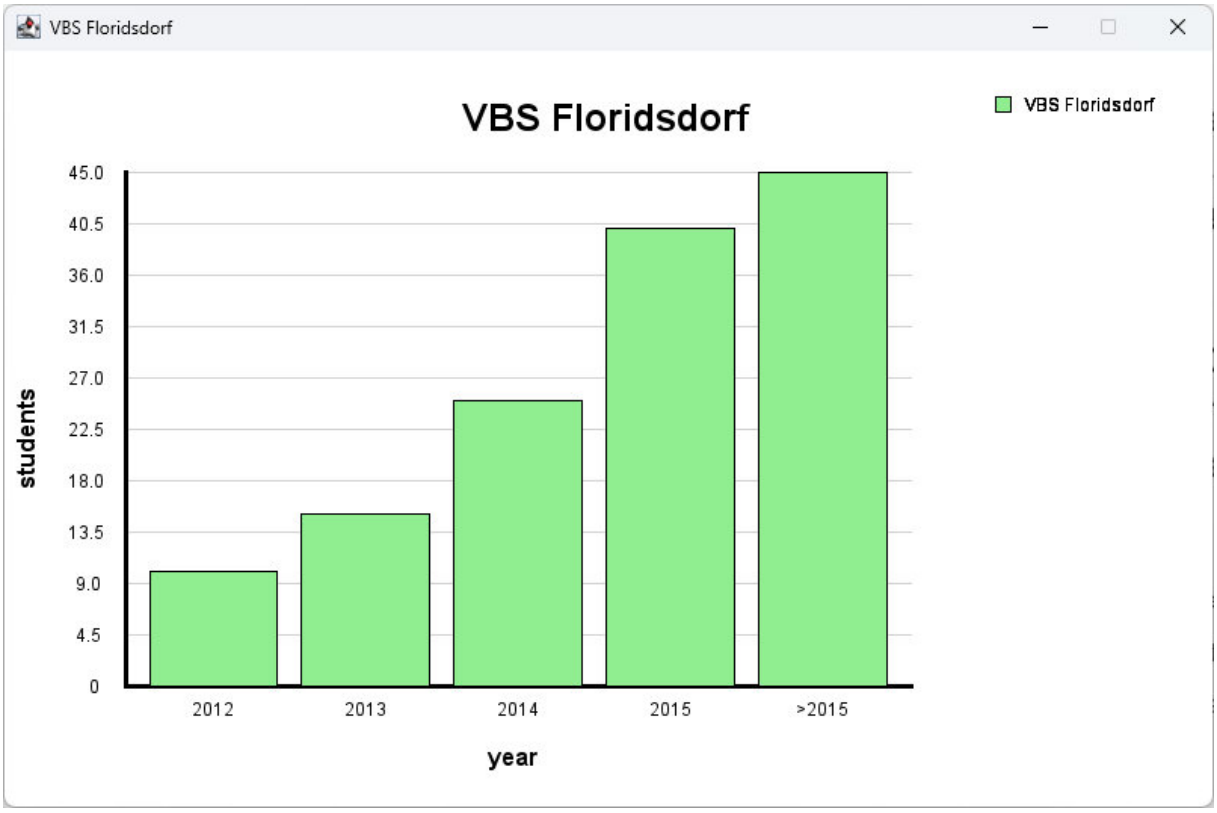

Figure 10: Output 01_column_chart.rxj

4.3. Bar Chart

The program “02_bar_chart.rxj” demonstrates the creation of a bar chart. The only differences to the previous example are the chart type argument in line 10 and some cosmetic changes. In line 12 the background color is changed and in line 13 the attribute

"legend?" is set to false, resulting in the legend being hidden once the chart is drawn. The "draw" method is used to display the bar chart, as shown in Figure 11.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
8
9  /* creating classic bar chart & display it */
10 myBarChart = .JChart~createChart("BarChart", myData, "VBS Meidling", "students", "year")
11                                   --chartType, dataSet [, title, xLabel, yLabel, width, height]
12 myBarChart~background = silver          --change background color to "silver" (default: "white")
13 myBarChart~legend? = .false             --disable the legend
14 myBarChart~draw
15
16 /* close windows after 20 seconds */
17 say "waiting 20 seconds"
18 call syssleep 20
19
20 /* get access to JBusinessChart classes */
21 ::requires 'JBusinessChart.cls'
```

Listing 6: 02_bar_chart.rxj

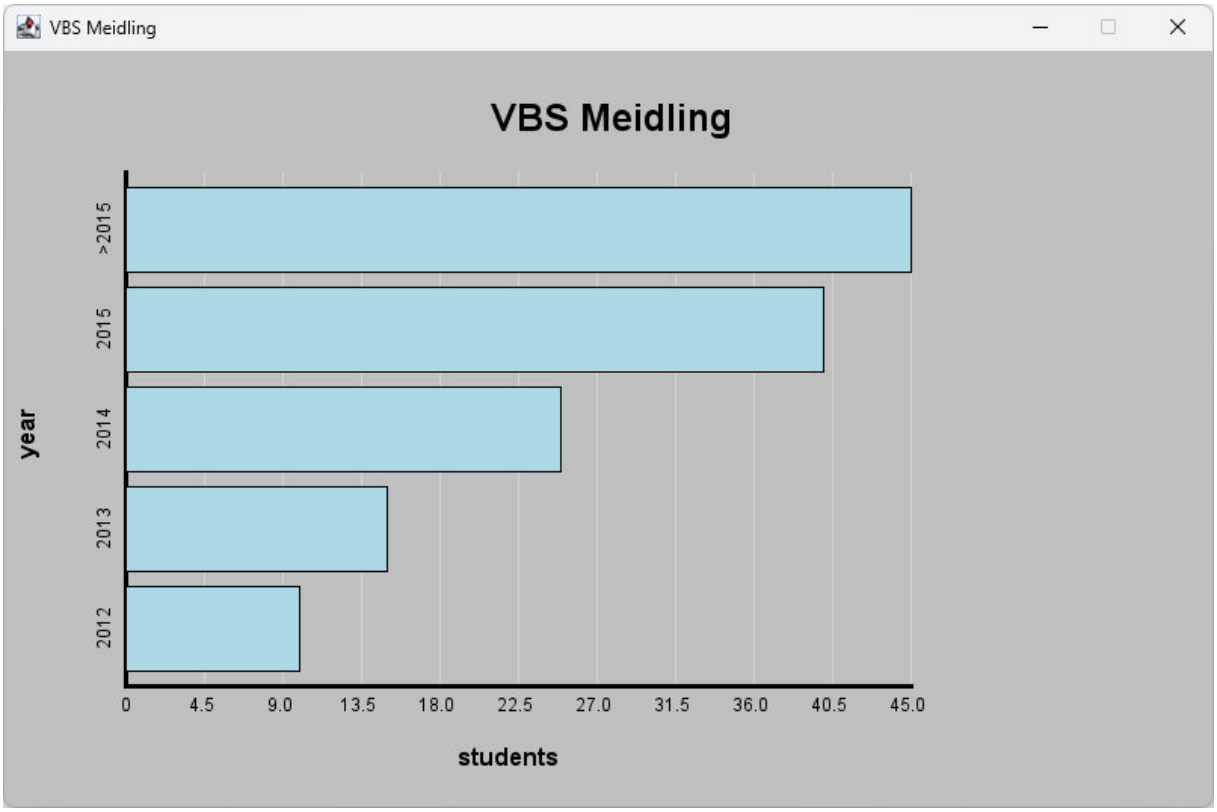

Figure 11: Output 02_bar_chart.rxj

4.4. Line Chart

The fourth example focuses on the "LineChart" class. As with the previous nutshell examples, an instance of the "DataSet" class is created. In line 10 of Listing 7, the line chart is assigned

the nickname "myLineChart". Lines 12 and 13 demonstrate different options for visual appearance. First, the default visible area below the line is disabled and in the next line, the line format is changed. As described in line 13 of Listing 7, by default the line is solid. Setting the value to 1 makes it dashed, 2 changes it to dotted and 3 results is a mix of dashed and dotted. The generated line chart is displayed in Figure 12.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
8
9  /* creating classic line chart & display it */
10 myLineChart = .JChart~createChart("LineChart", myData, "VBS Akademiestraße", "year", "students")
11                                    --chartType, dataSet [, title, xLabel, yLabel, width, height]
12 myLineChart~area? = .false      --disable showing area under the line (default: .true)
13 myLineChart~lineFormat = 1    --change line format (default: 0) [0: line, 1: dashed, 2: dotted, 3: mixed]
14 myLineChart~draw
15
16 /* close windows after 20 seconds */
17 say "waiting 20 seconds"
18 call syssleep 20
19
20 /* get access to JBusinessChart classes */
21 ::requires 'JBusinessChart.cls'
```

Listing 7: 03_line_chart.rxj

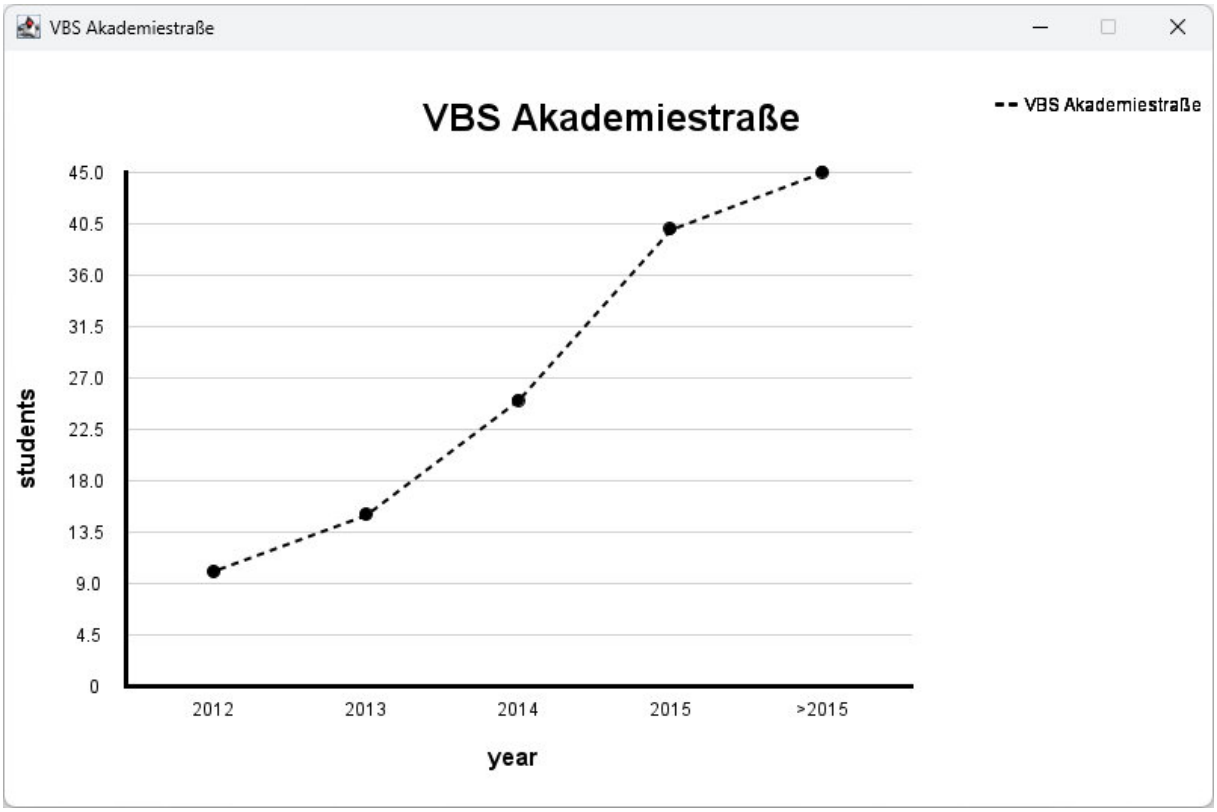

Figure 12: Output 03_line_chart.rxj

4.5. Point Chart

This nutshell example will generate a point chart. Once again, a dataset is created, however, this time both the x and y values are numeric. This small but significant change is crucial, as otherwise an error would occur. In line 12 of Listing 8, the attribute “point?” is set to false, which results in hiding the points in the generated chart, as seen in Figure 13.

As a brief recap, other cosmetic options are possible, such as changing the color. For simplicity, only a handful of options are demonstrated in each example.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue(0, 10)
4  myData~addValue(10, 15)
5  myData~addValue(25, 20)
6  myData~addValue(30, 30)
7  myData~addValue(50, 12)
8
9  /* creating classic point chart & display it */
10 myPointChart = .JChart~createChart("PointChart", myData, "Students & Teachers", "students", "teachers")
11                                    --chartType, dataSet [, title, xLabel, yLabel, width, height]
12 myPointChart~points? = .false     --disable showing points (default: .true)
13 myPointChart~draw
14
15 /* close windows after 20 seconds */
16 say "waiting 20 seconds"
17 call syssleep 20
18
19 /* get access to JBusinessChart classes */
20 ::requires 'JBusinessChart.cls'
```

Listing 8: 04_point_chart.xrxj

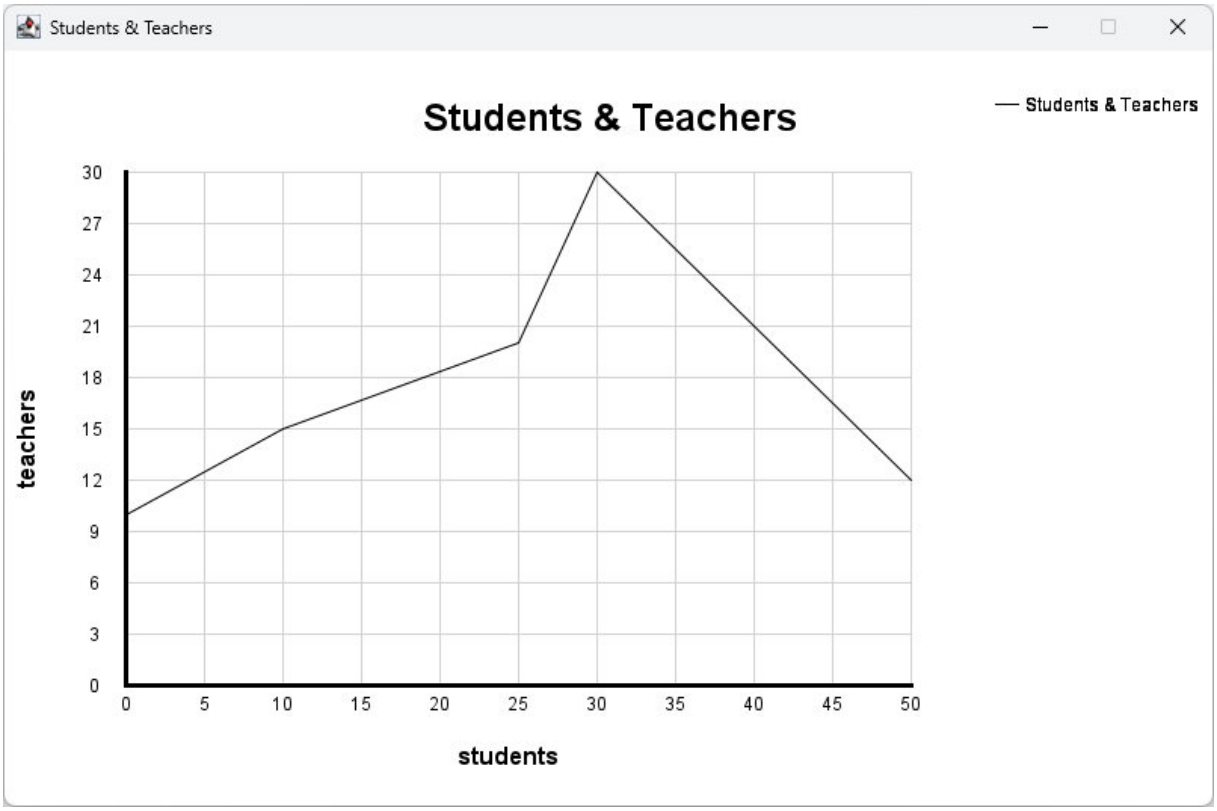

Figure 13: Output 04_point_chart.rxj

4.6. Pie Chart

The example "05_pie_chart.rxj" demonstrates the use of the "PieChart" class. Up until line 9 of Listing 9, the process is identical to that of the nutshell examples "01_column_chart.rxj" and "02_bar_chart.rxj". To create a pie chart, the String used for the chart type argument must be "PieChart". This chart type has fewer optional arguments, as shown in line 11. The possibility to change the colors of the segments using an array is demonstrated in line 12. The final result is displayed in Figure 14.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
8
9  /* creating classic pie chart & display it */
10 myPieChart = .JChart~createChart("PieChart", myData, "VBS Floridsdorf")
11                                   --chartType, dataSet [, title, width, height]
12 myPieChart~colorPallet = ("lightblue", "cyan", "yellow", "orange", "red", "blue")
13  --change color of elements (default: random)
14 myPieChart~draw
15
16 /* close windows after 20 seconds */
17 say "waiting 20 seconds"
18 call syssleep 20
19
```

```
20 /* get access to JBusinessChart classes */
21 ::requires 'JBusinessChart.cls'
```

Listing 9: 05_pie_chart.rxj

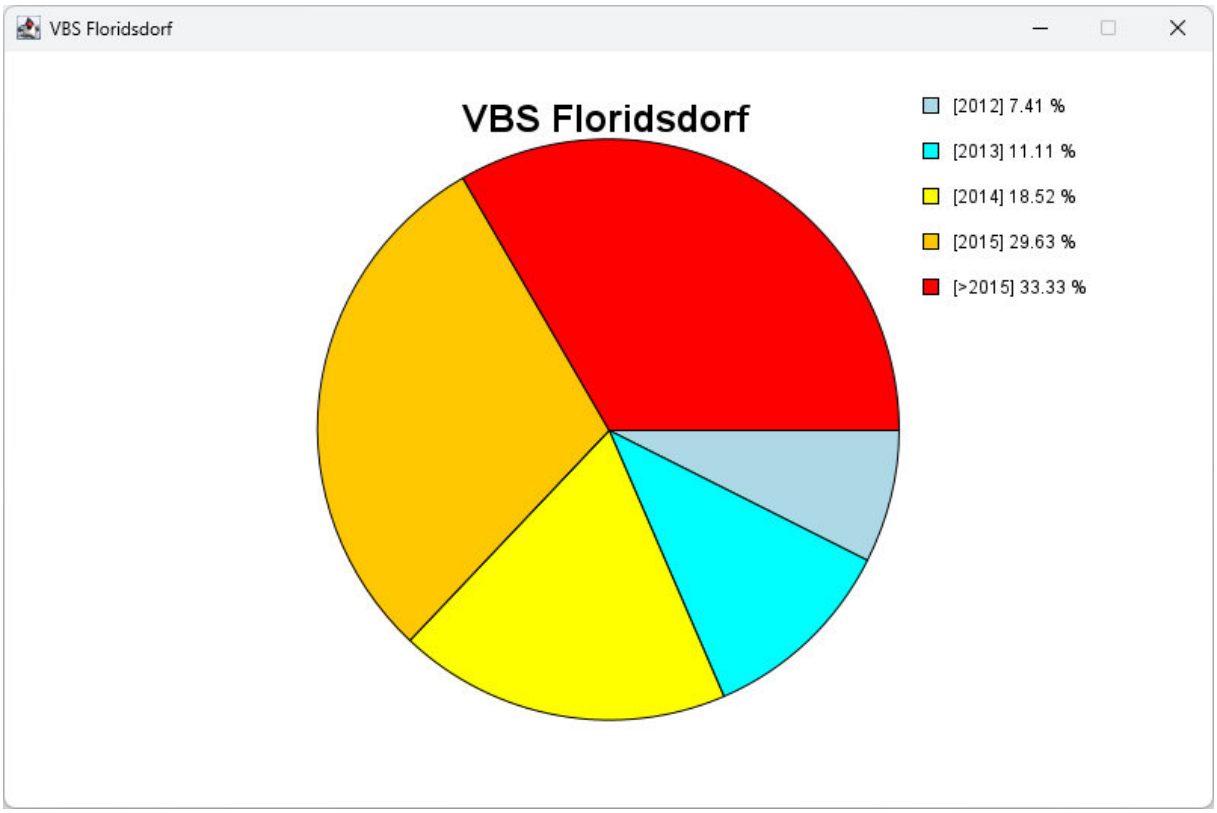

Figure 14: Output 05_pie_chart.rxj

4.7. Ring Chart

The final nutshell example of this difficulty level is "06_ring_chart.rxj". This program generates a ring chart, using a structure very similar to the previous example. This time, no custom color pallet is provided. Instead, the colors are randomly generated. Additionally, the optional "width" and "height" arguments of the "createChart" method in line 10 of Listing 10 are modified. This results in the chart's dimensions being changed, as shown in Figure 15. To complete the chart, the "unit" attribute for the total amount is defined. Otherwise, "units" would be displayed.

```
1  /* creating datasets */
2  myData = .dataSet~new
3  myData~addValue("2012", 10)
4  myData~addValue("2013", 15)
5  myData~addValue("2014", 25)
6  myData~addValue("2015", 40)
7  myData~addValue(">2015", 45)
8
9  /* creating classic ring chart & display it */
10 myRingChart = .JChart~createChart("RingChart", myData, "VBS Meidling", , , 1000, 800)
11                                  --chartType, dataSet [, title, , ,width, height]
12 myRingChart~unit = "Students"        --changing unit of the data displayed in the middle
13 myRingChart~draw
```

```
14
15 /* close windows after 20 seconds */
16 say "waiting 20 seconds"
17 call syssleep 20
18
19 /* get access to JBusinessChart classes */
20 ::requires 'JBusinessChart.cls'
```

Listing 10: 06_ring_chart.rxj

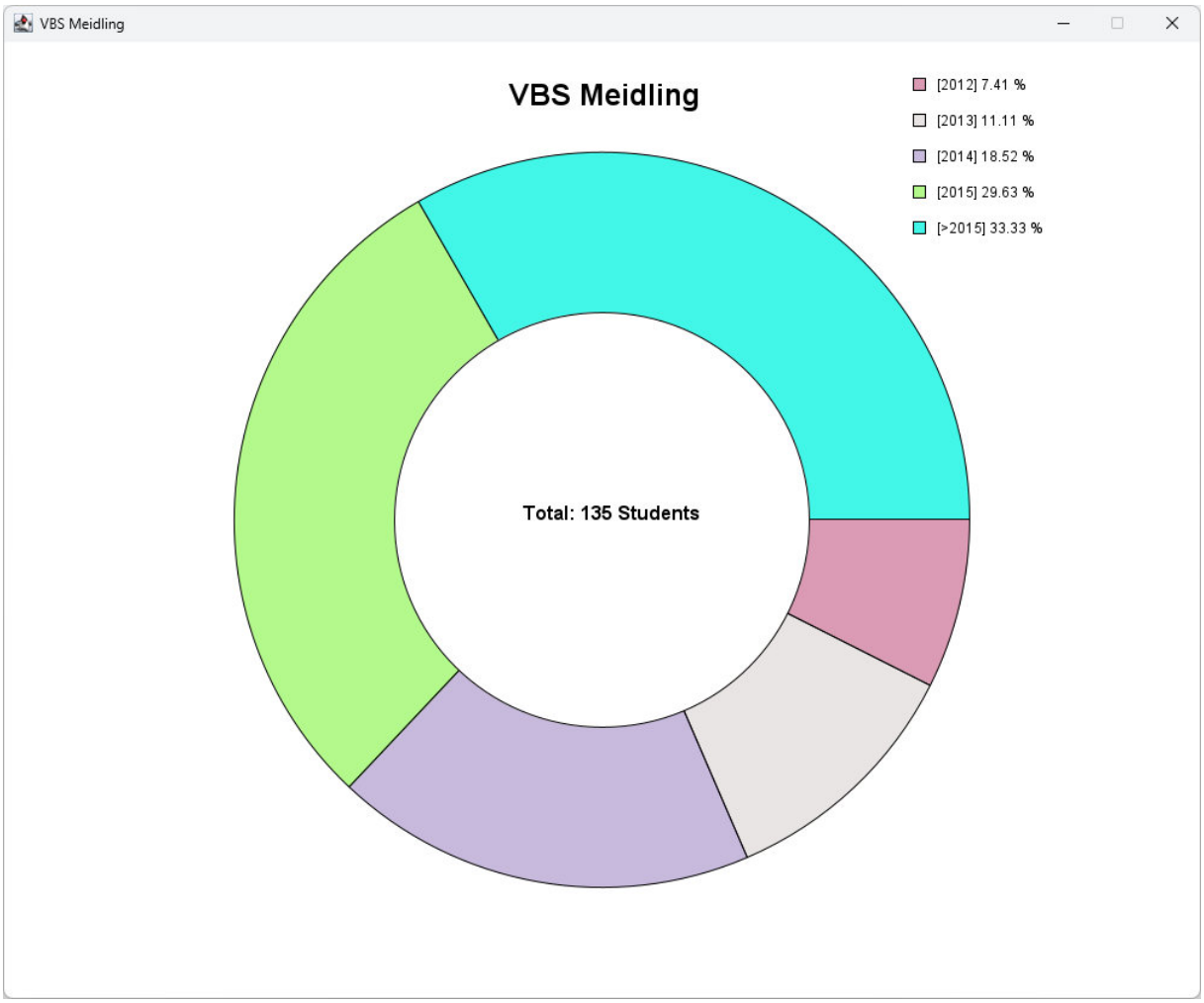

Figure 15: Output 06_ring_chart.rxj

4.8. Combine Column Charts

The combine method of the "JChart" class has two functions. The first function, which combines multiple charts of the same type into a single "MultiChart" object, is demonstrated by the following program. First, three different datasets representing the number of students in a school are created, numbered from one to three. These datasets are then used to instantiate three separate column charts. To enhance visibility differences, in lines 37 and 28 of Listing 11, each chart is assigned its own color. In line 31, the charts are combined and the result is assigned the name "myCombinedColumns". The "draw" method then displays the business chart, as Figure 16 shows.

```
/* creating datasets */
hak1 = .dataSet~new
hak1~addValue("2012", 10)
hak1~addValue("2013", 15)
hak1~addValue("2014", 25)
hak1~addValue("2015", 40)
hak1~addValue(">2015", 45)

hak2 = .dataSet~new
hak2~addValue("2012", 10)
hak2~addValue("2013", 30)
hak2~addValue("2014", 20)
hak2~addValue("2015", 25)
hak2~addValue(">2015", 30)

hak3 = .dataSet~new
hak3~addValue("2012", 30)
hak3~addValue("2013", 40)
hak3~addValue("2014", 35)
hak3~addValue("2015", 25)
hak3~addValue(">2015", 20)

/* creating column charts */
c1 = .JChart~createChart("ColumnChart", hak1, "VBS Floridsdorf", "year", "students")
c2 = .JChart~createChart("ColumnChart", hak2, "VBS Meidling", "year", "students")
c3 = .JChart~createChart("ColumnChart", hak3, "VBS Akademiestraße", "year", "students")
c2~color = red
c3~color = lightgreen

/* combining column charts */
myCombinedColumns = .JChart~combineCharts(c1, c2, c3)
myCombinedColumns~title = "Student Number Development"
myCombinedColumns~draw

/* close windows after 20 seconds */
say "waiting 20 seconds"
call syssleep 20

/* get access to JBusinessChart classes */
::requires 'JBusinessChart.cls'
```

Listing 11: 11_combine_column_charts.rxj

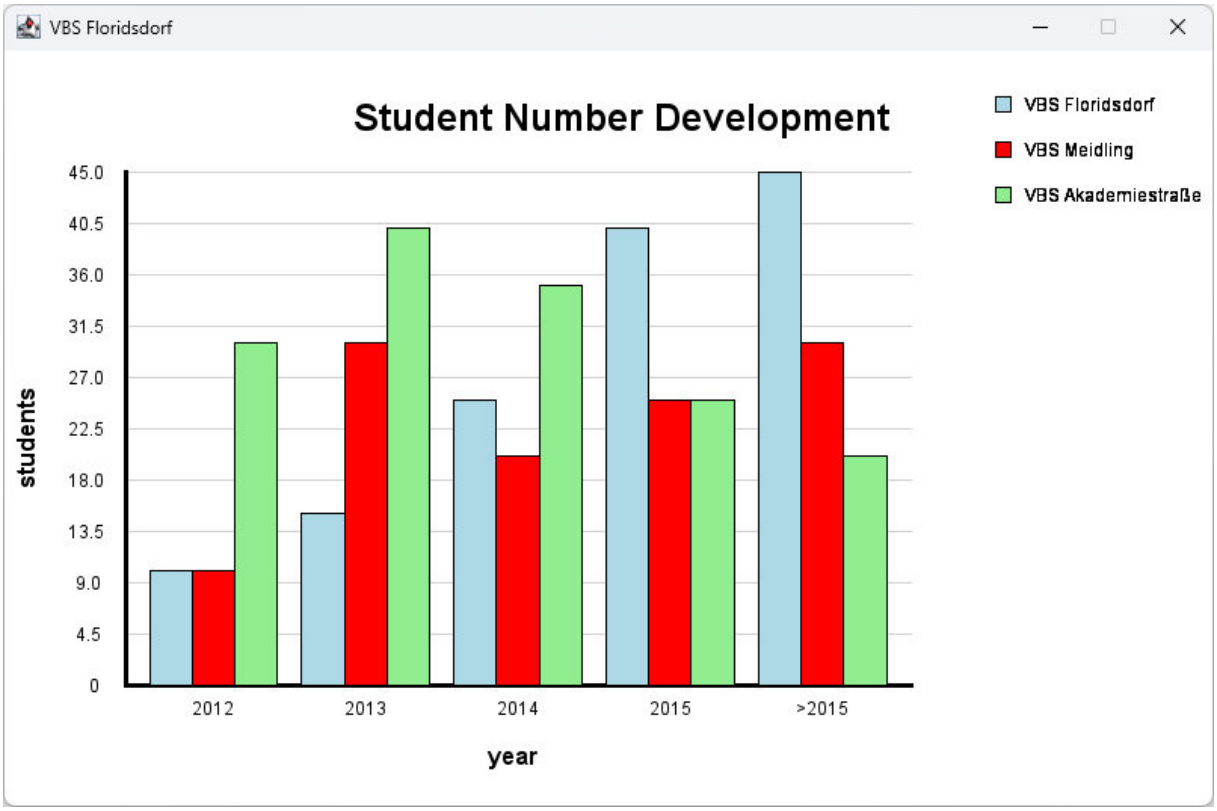

Figure 16: Output 11_combine_column_charts.rxj

4.9. Combine Line Charts

The same process works also with other chart types, as demonstrated in this nutshell example with line charts. Again, the visual appearance is adjusted to enhance visibility differences. This is especially important for the legend, because, as Figure 17 shows, it allows the viewer to easily connect the legend with the corresponding lines in the chart.

```
1  /* creating datasets */
2  hak1 = .dataSet~new
3  hak1~addValue("2012", 10)
4  hak1~addValue("2013", 15)
5  hak1~addValue("2014", 25)
6  hak1~addValue("2015", 40)
7  hak1~addValue(">2015", 45)
8
9  hak2 = .dataSet~new
10 hak2~addValue("2012", 10)
11 hak2~addValue("2013", 30)
12 hak2~addValue("2014", 20)
13 hak2~addValue("2015", 25)
14 hak2~addValue(">2015", 30)
15
16 /* creating line charts */
17 l1 = .JChart~createChart("LineChart", hak1, "VBS Floridsdorf", "year", "students")
18 l2 = .JChart~createChart("LineChart", hak2, "VBS Meidling", "year", "students")
19 l1~color = red
20 l1~area? = .false
21 l2~area? = .false
22 l2~lineFormat = 3
23
```

```
24  /* combining line charts */
25  myCombinedLines = .JChart~combineCharts(l1, l2)
26  myCombinedLines~title = "Student Number Development"
27  myCombinedLines~draw
28
29  /* close windows after 20 seconds */
30  say "waiting 20 seconds"
31  call syssleep 20
32
33  /* get access to JBusinessChart classes */
34  ::requires 'JBusinessChart.cls'
```

Listing 12: 12_combine_line_charts.rxj

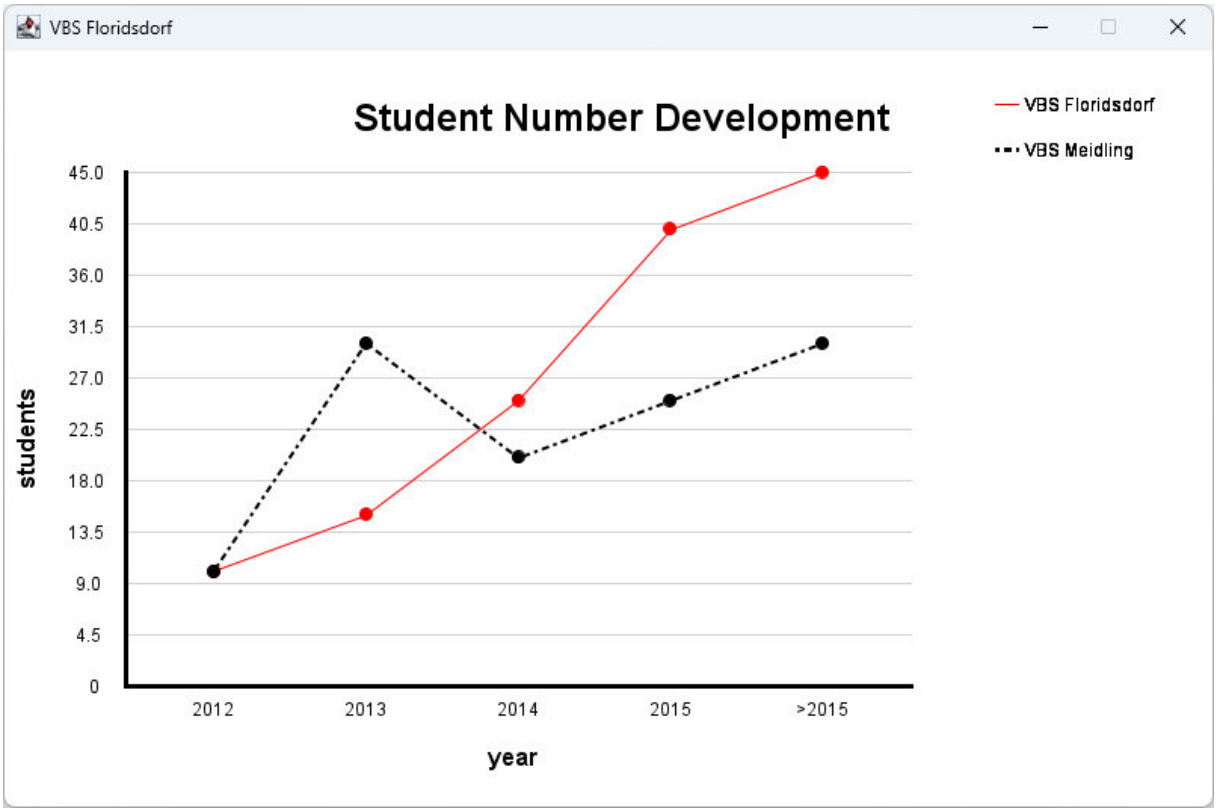

Figure 17: Output 12_combine_line_charts.rxj

4.10. Mixing Column and Line Charts

As mentioned in section 4.8., the combine method of the "JChart" class has two functions. This nutshell example demonstrates the second functionality: combining or rather mixing a column and line chart together. This process begins by creating imaginary datasets of schools and generating corresponding charts from them. By using the "combineCharts" method, as shown in line 26 of Listing 13, and providing the chart objects as arguments, the "JChart" class knows that it needs to create a mixed chart. As always, by sending the "draw" message, the chart is generated with the result in Figure 18.

```
1  /* creating datasets */
2  hak1 = .dataSet~new
3  hak1~addValue("2012", 10)
4  hak1~addValue("2013", 15)
```

```
 5  hak1~addValue("2014", 25)
 6  hak1~addValue("2015", 40)
 7  hak1~addValue(">2015", 45)
 8
 9  hak3 = .dataSet~new
10  hak3~addValue("2012", 30)
11  hak3~addValue("2013", 40)
12  hak3~addValue("2014", 35)
13  hak3~addValue("2015", 25)
14  hak3~addValue(">2015", 20)
15
16  /* creating column chart */
17  c3 = .JChart~createChart("ColumnChart", hak3, "VBS Akademiestraße", "year", "students")
18  c3~color = lightgreen
19
20  /* creating line chart */
21  l1 = .JChart~createChart("LineChart", hak1, "VBS Floridsdorf", "year", "students")
22  l1~color = red
23  l1~area? = .false
24
25  /* combining different chart types */
26  myMix1 = .JChart~combineCharts(c3, l1) --line chart and column chart order does not matter
27  myMix1~title = "Student Number Development"
28  myMix1~draw
29
30  /* close windows after 20 seconds */
31  say "waiting 20 seconds"
32  call syssleep 20
33
34  /* get access to JBusinessChart classes */
35  ::requires 'JBusinessChart.cls'
```

Listing 13: 13_mix_column-line_charts.rxj

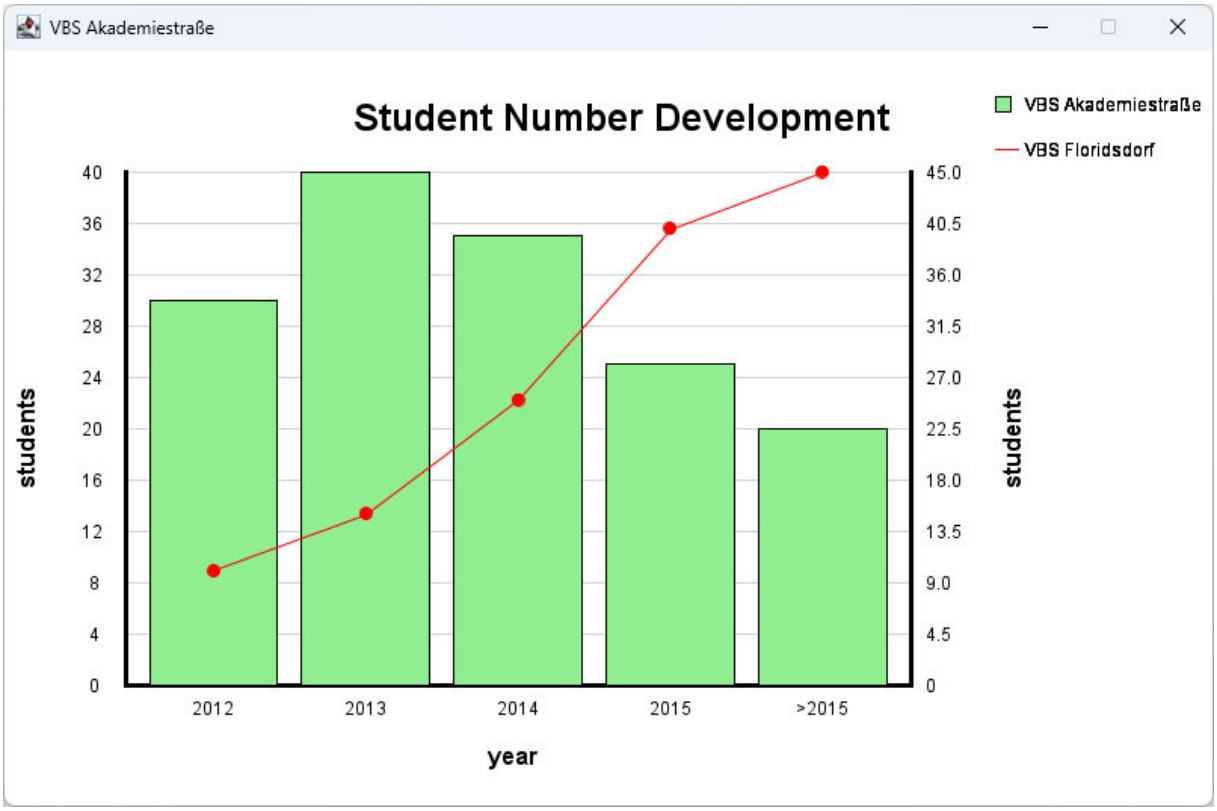

Figure 18: Output 13_mix_column-line_charts.rxj

4.11. Mixing Combined Charts

The mixing feature of the "combineCharts" methods works also with instances of the "MultiChart" class. This means that the resulting chart objects from sections 4.8. and 4.9. can also be mixed, similar to the process in the nutshell example above. This is achieved by first creating the individual charts, combining them and then combining/mixing the result again, as shown in line 41 of Listing 14. The final is illustrated in Figure 19.

```
1  /* creating datasets */
2  hak1 = .dataSet~new
3  hak1~addValue("2012", 10)
4  hak1~addValue("2013", 15)
5  hak1~addValue("2014", 25)
6  hak1~addValue("2015", 40)
7  hak1~addValue(">2015", 45)
8
9  hak2 = .dataSet~new
10 hak2~addValue("2012", 10)
11 hak2~addValue("2013", 30)
12 hak2~addValue("2014", 20)
13 hak2~addValue("2015", 25)
14 hak2~addValue(">2015", 30)
15
16 hak3 = .dataSet~new
17 hak3~addValue("2012", 30)
18 hak3~addValue("2013", 40)
19 hak3~addValue("2014", 35)
20 hak3~addValue("2015", 25)
21 hak3~addValue(">2015", 20)
22
23 /* creating column charts */
24 c1 = .JChart~createChart("ColumnChart", hak1, "VBS Floridsdorf", "year", "students")
25 c2 = .JChart~createChart("ColumnChart", hak2, "VBS Meidling", "year", "students")
26 c3 = .JChart~createChart("ColumnChart", hak3, "VBS Akademiestraße", "year", "students")
27 c2~color = red
28 c3~color = lightgreen
29
30 /* combining column charts */
31 myCombinedColumns = .JChart~combineCharts(c1, c2, c3)
32
33 /* creating line charts */
34 l1 = .JChart~createChart("LineChart", hak1, "VBS Floridsdorf", "year", "students")
35 l2 = .JChart~createChart("LineChart", hak2, "VBS Meidling", "year", "students")
36 l1~color = red
37 l1~area? = .false
38 l2~area? = .false
39 l2~lineFormat = 3
40
41 /* combining line charts */
42 myCombinedLines = .JChart~combineCharts(l1, l2)
43
44 /* combining different combined chart types */
45 myMix2 = .JChart~combineCharts(myCombinedColumns, myCombinedLines)
46 myMix2~title = "Student Number Development"
47 myMix2~draw
48
49 /* close windows after 20 seconds */
50 say "waiting 20 seconds"
51 call syssleep 20
52
```

```
53 /* get access to JBusinessChart classes */
54 ::requires 'JBusinessChart.cls'
```

Listing 14: 14_mix_combined_charts.rxj

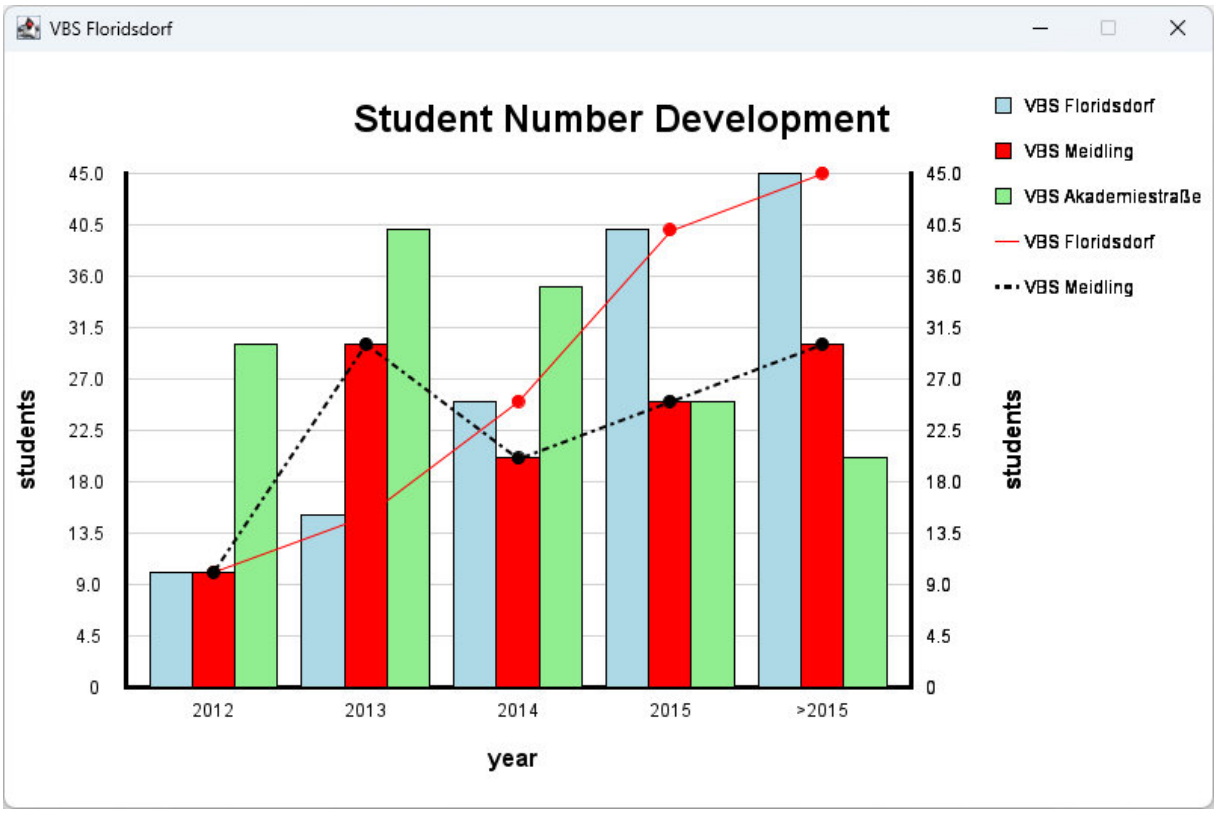

Figure 19: Output 14_mix_combined_charts.rxj

4.12. Stack Bar Charts

The nutshell example "21_stack_bar_charts.rxj" demonstrates the use of the "stackCharts" method of the "JChart" class. Similar to the "combineCharts" method, the "stackCharts" method can work with different chart types. In this example, bar charts are stacked on top of each other to create a "StackChart" object, as illustrated in line 31 of Listing 15. Prior to this, datasets are created (lines 1 to 21) and bar charts are instantiated and customized (lines 23 to 28). The resulting chart is displayed in Figure 20.

```
1  /* creating datasets */
2  hak1 = .dataSet~new
3  hak1~addValue("2012", 10)
4  hak1~addValue("2013", 15)
5  hak1~addValue("2014", 25)
6  hak1~addValue("2015", 40)
7  hak1~addValue(">2015", 45)
8
9  hak2 = .dataSet~new
10 hak2~addValue("2012", 10)
11 hak2~addValue("2013", 30)
12 hak2~addValue("2014", 20)
13 hak2~addValue("2015", 25)
14 hak2~addValue(">2015", 30)
15
```

```
16 hak3 = .dataSet~new
17 hak3~addValue("2012", 30)
18 hak3~addValue("2013", 40)
19 hak3~addValue("2014", 35)
20 hak3~addValue("2015", 25)
21 hak3~addValue(">2015", 20)
22
23 /* create bar charts */
24 b1 = .JChart~createChart("BarChart", hak1, "VBS Floridsdorf", "students", "year")
25 b2 = .JChart~createChart("BarChart", hak2, "VBS Meidling", "students", "year")
26 b3 = .JChart ~createChart("BarChart", hak3, "VBS Akademiestraße", "students", "year")
27 b2~color = turquoise
28 b3~color = pink
29
30 /* stacking bar charts */
31 myStackedBars = .JChart~stackCharts(b1, b2, b3)
32 myStackedBars~title = "Student Number Development"
33 myStackedBars~draw
34
35 /* close windows after 20 seconds */
36 say "waiting 20 seconds"
37 call syssleep 20
38
39 /* get access to JBusinessChart classes */
40 ::requires 'JBusinessChart.cls'
```

Listing 15: 21_stack_bar_charts.rxj

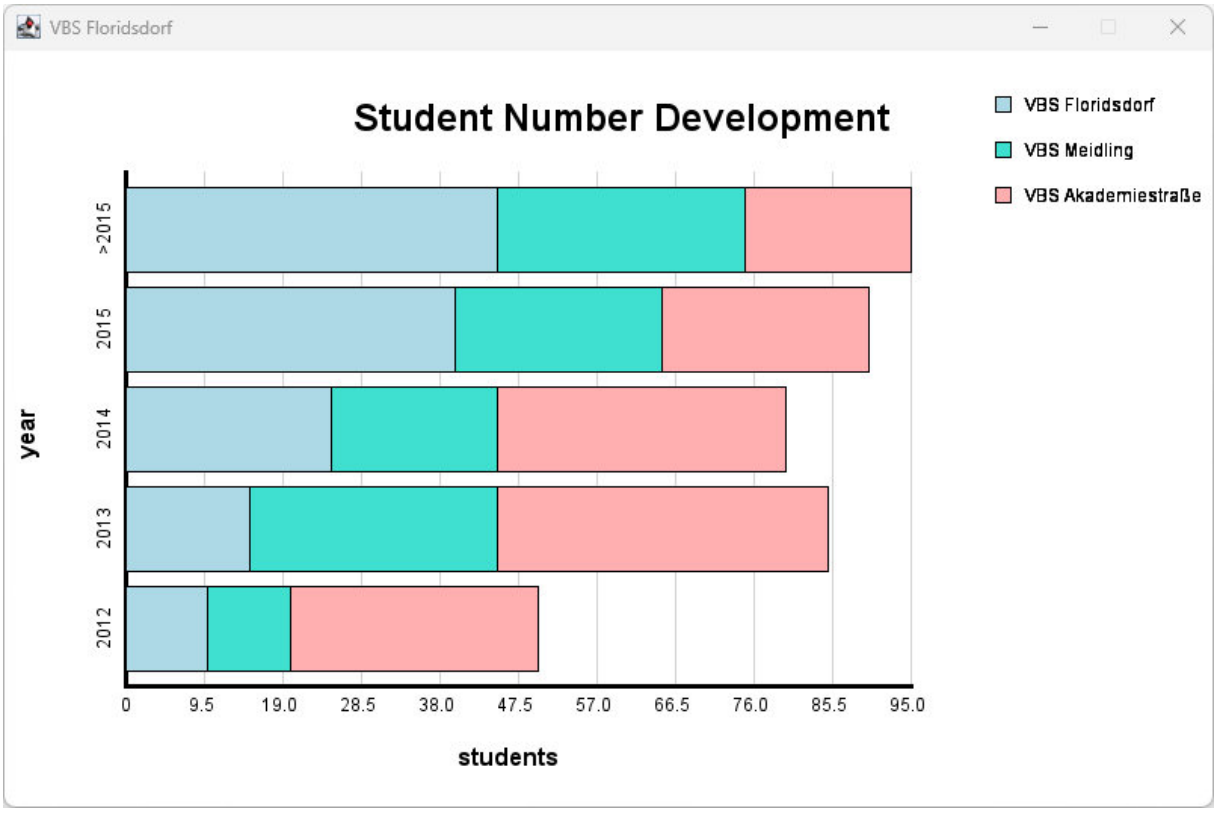

Figure 20: Output 21_stack_bar_charts.rxj

4.13.Stack Line Charts

The same creation process applies for a stacked line chart. To enhance visibility, the areas below the lines are filled with the line color in this example, allowing for clearer representation of the stacked lines.

```
/* creating datasets */
hak1 = .dataSet~new
hak1~addValue("2012", 10)
hak1~addValue("2013", 15)
hak1~addValue("2014", 25)
hak1~addValue("2015", 40)
hak1~addValue(">2015", 45)

hak2 = .dataSet~new
hak2~addValue("2012", 10)
hak2~addValue("2013", 30)
hak2~addValue("2014", 20)
hak2~addValue("2015", 25)
hak2~addValue(">2015", 30)

hak3 = .dataSet~new
hak3~addValue("2012", 30)
hak3~addValue("2013", 40)
hak3~addValue("2014", 35)
hak3~addValue("2015", 25)
hak3~addValue(">2015", 20)

/* creating line charts */
l1 = .JChart~createChart("LineChart", hak1, "VBS Floridsdorf", "year", "students")
l2 = .JChart~createChart("LineChart", hak2, "VBS Meidling", "year", "students")
l3 = .JChart~createChart("LineChart", hak3, "VBS Akademiestraße", "year", "students")
l1~color = blue
l2~color = red
l3~color = green

/* stacking line charts */
myStackedLines = .JChart~stackCharts(l1, l2, l3)

myStackedLines~draw
myStackedLines~title = "Student Number Development"
/* close windows after 20 seconds */
say "waiting 20 seconds"
call syssleep 20

/* get access to JBusinessChart classes */
::requires 'JBusinessChart.cls'
```

Listing 16: 22_stack_line_charts.rxj

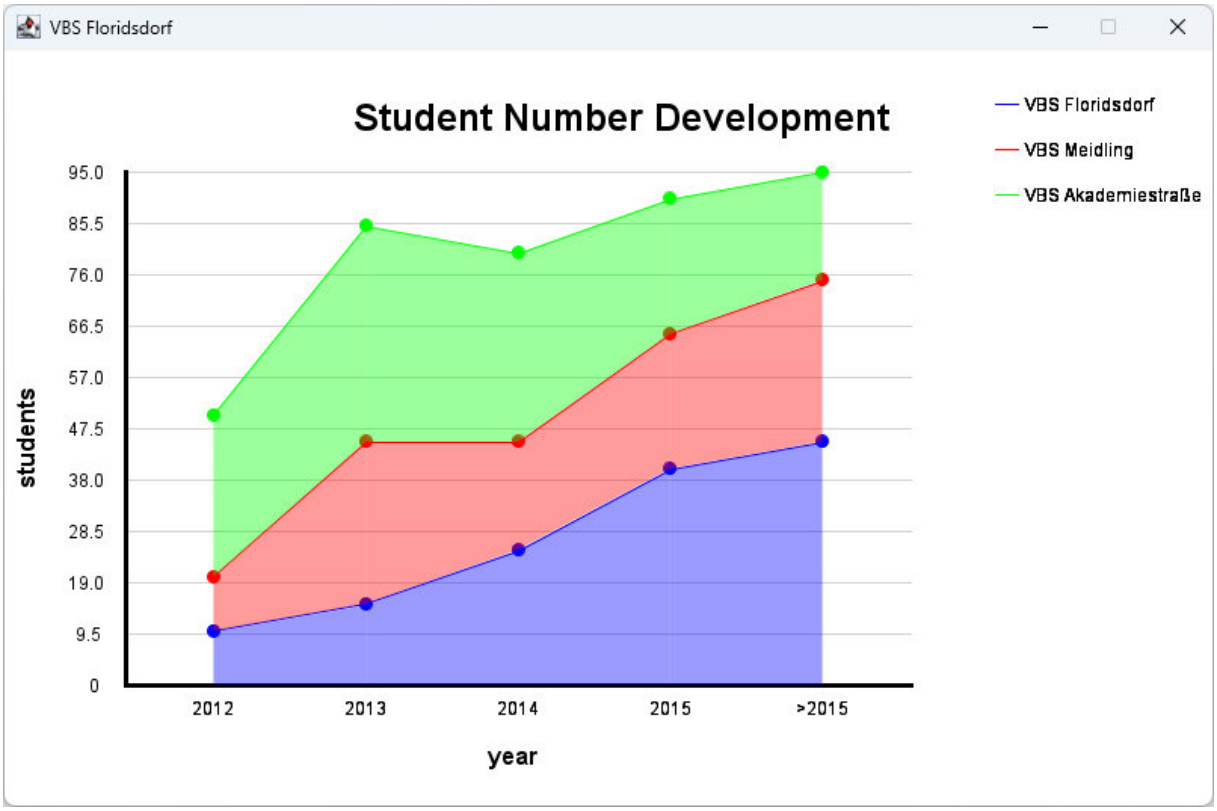

Figure 21: Output 22_stack_line_charts.rxj

4.14. Stack Charts in Percent

The very last nutshell example mirrors the process described in section 4.13. The key difference is in line 35 of Listing 17. The attribute "toPercent" is changed from the default value "false" to "true". This modification alters the representation of the data to percentages, resulting in the y-axis displaying percentages, as shown in Figure 22.

```
1  /* creating datasets */
2  hak1 = .dataSet~new
3  hak1~addValue("2012", 10)
4  hak1~addValue("2013", 15)
5  hak1~addValue("2014", 25)
6  hak1~addValue("2015", 40)
7  hak1~addValue(">2015", 45)
8
9  hak2 = .dataSet~new
10 hak2~addValue("2012", 10)
11 hak2~addValue("2013", 30)
12 hak2~addValue("2014", 20)
13 hak2~addValue("2015", 25)
14 hak2~addValue(">2015", 30)
15
16 hak3 = .dataSet~new
17 hak3~addValue("2012", 30)
18 hak3~addValue("2013", 40)
19 hak3~addValue("2014", 35)
20 hak3~addValue("2015", 25)
21 hak3~addValue(">2015", 20)
22
23 /* creating line charts */
```

```
24 | l1 = .JChart~createChart("LineChart", hak1, "VBS Floridsdorf", "year", "students")
25 | l2 = .JChart~createChart("LineChart", hak2, "VBS Meidling", "year", "students")
26 | l3 = .JChart~createChart("LineChart", hak3, "VBS Akademiestraße", "year", "students")
27 | l1~color = blue
28 | l2~color = red
29 | l3~color = green
30 |
31 | /* stacking line charts */
32 | myStackedLines = .JChart~stackCharts(l1, l2, l3)
33 | myStackedLines~title = "Student Number Development"
34 |
35 | /* change data representation to 100 percent*/
36 | myStackedLines~toPercent = .true
37 | myStackedLines~draw
38 |
39 | /* close windows after 20 seconds */
40 | say "waiting 20 seconds"
41 | call syssleep 20
42 |
43 | /* get access to JBusinessChart classes */
44 | ::requires 'JBusinessChart.cls'
```

Listing 17: 23_stack_charts_in_percent.rxj

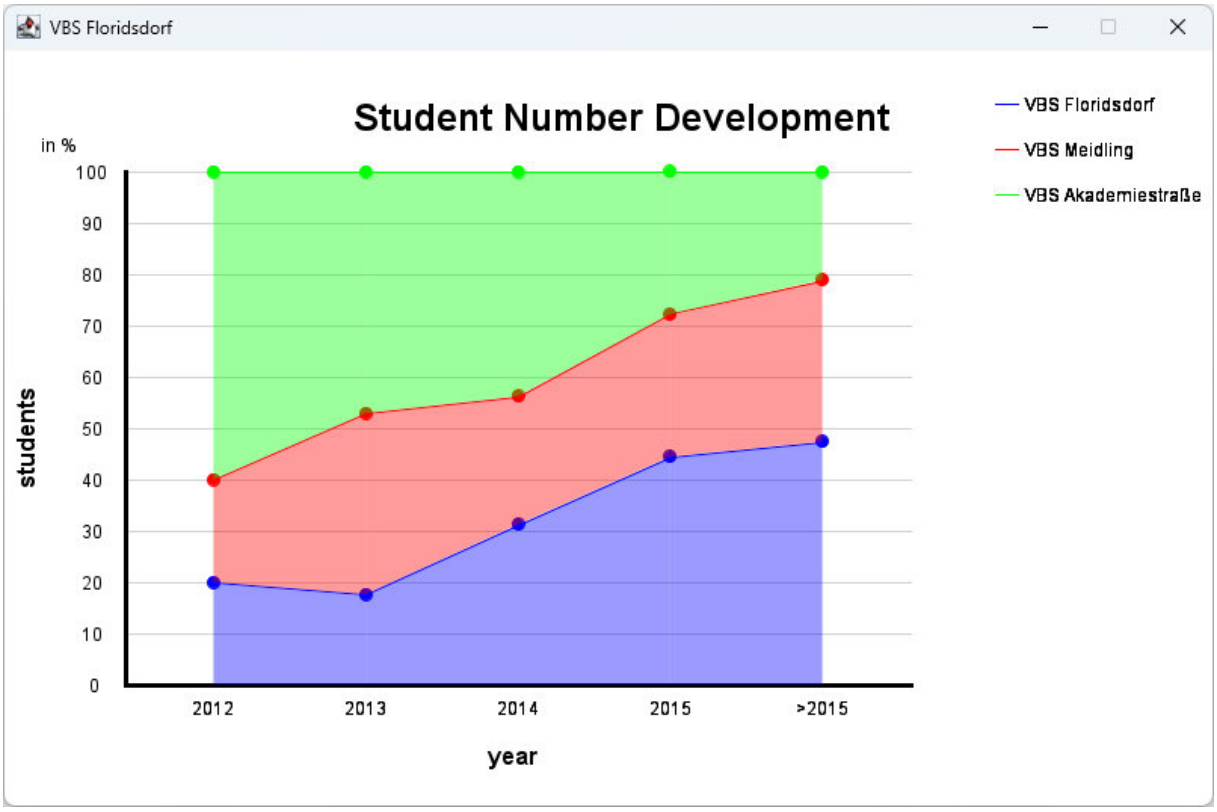

Figure 22: Output 23_stack_charts_in_percent.rxj

5. Brief Discussion

JBusinessChart offers extensive capabilities for business chart creation, but it is important to recognize its limitations as an initial version. A key limitation is that it does not support the representation of negative values.

Also, a debatable point is that many methods do not enforce restrictions on the user input. JBusinessChart relies on the user's common knowledge and familiarity with the framework, which could lead to errors. Although input validation could be implemented, such checks were intentionally omitted. During development and expansion, this unrestricted approach led to the discovery of new functionalities. For example, it became possible to mix “MultiChart” objects made from column and line charts, a feature initially unintended but later incorporated. This framework aims to foster creativity and curiosity, allowing users to explore additional options that may be introduced in future versions. Additional work, such as implementing error codes, could help balance these considerations by supporting users in programming while still providing flexibility.

Currently, JBusinessChart offers a diverse range of business chart types comparable to those available in Microsoft Excel. However, the range of chart types provided by JBusinessChart is not exhaustive and there is potential to implement new types in the future.

Furthermore, JBusinessChart's architecture could be improved. The continuous expansion and addition of new chart types and features have led to a complex and tightly intertwined codebase, making further expansion challenging or, at times, nearly impossible. The distribution of tasks for canvas creation across the "Canvas" and "CoorSys" classes is, in retrospect, questionable. Additionally, the approach used for generating the “StackChart” creates redundancy, which was necessary due to the lack of initial planning. Small changes are difficult to implement because the code is tightly interwoven, a problem that became already apparent during development. Rewriting the code may be necessary in the future to address these issues and improve modularity.

6. Conclusion

In this thesis, JBusinessChart is developed and introduced. JBusinessChart is a framework, built on top of JDOR, a command handler for Java2D integrated within the BSF4ooRexx package, which simplifies the creation of business charts. This significantly facilitates and improves the visual representation of data for ooRexx programmers.

One of the key contributions of JBusinessCharts to ooRexx is making business charts available to all programmers, even those with little to no prior knowledge of JDOR or Java2D. However, additional knowledge of JDOR can expand the capabilities of JBusinessChart. With the user-friendly approach of JBusinessChart and the nutshell examples, users can create professional charts in a short amount of time. An important characteristic of this framework is that despite the simplicity of its handling, JBusinessChart offers considerable customization options for the appearance of business charts, imposing minimal restrictions on the user.

Looking ahead, JBusinessChart has the potential to become even more powerful by expanding the range of available chart types and incorporating support for negative numbers. Additionally, rewriting the existing codebase would be beneficial to facilitate future development.

References

[1] "Open Object Rexx," Rexx Language Association, [Online]. Available: https://www.oorexx.org/about.html. [Accessed 24 July 2024].

[2] R. G. Flatscher, "JDOR - Java2D for ooRexx (and Other Programming Languages)," in *2024 International Rexx Language Symposium Proceedings*, R. V. Jansen, Ed., Amsterdam, Rexx Language Association, 2024, pp. 88-98.

[3] "BSF4ooRexx download | SourceForge.net," SourceForge, 7 July 2024. [Online]. Available: https://sourceforge.net/projects/bsf4oorexx/. [Accessed 24 July 2024].

[4] "JDOR Synopsis," [Online]. Available: https://wi.wu.ac.at/rgf/rexx/misc/jdor_doc.tmp/jdor_doc.html. [Accessed 30 July 2024].

[5] "Java Download | Java 8, Java 11, Java 17, Java 21, Java 22 - OpenJDK Builds for Linux, Windows & macOS," bellsoft, [Online]. Available: https://bell-sw.com/pages/downloads/#jdk-22. [Accessed 24 July 2024].

[6] "Download oorexx-5.1.0-12859.windows.x86_64.exe (ooRexx (Open Object Rexx))," SourceForge, [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/oorexx-5.1.0-12859.windows.x86_64.exe/download. [Accessed 24 July 2024].

[7] "ooRexx (Open Object Rexx) - Browse / oorexx/5.1.0beta at SourceForge.net," SourceForge, [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/. [Accessed 12 August 2024].

[8] "Download BSF4ooRexx," SourceForge, [Online]. Available: https://sourceforge.net/projects/bsf4oorexx/files/latest/download. [Accessed 24 July 2024].

[9] "Download oorexx-5.1.0.12860.windows.x86_64-portable-release.zip (ooRexx (Open Object Rexx))," SourceForge, [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/portable/oorexx-5.1.0-12860.windows.x86_64-portable-release.zip/download. [Accessed 12 August 2024].

[10] "ooRexx (Open Object Rexx) - Brows /oorexx/5.1.0beta/portable at SourceForge.net," SourceForge, [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/portable/. [Accessed 12 August 2024].

[11] "What is Pascal Case? Definition, Types, and Examples.," Techopedia, [Online]. Available: https://www.techopedia.com/definition/pascal-case#:~:text=Pascal%20case%2C%20or%20PascalCase%2C%20is,or%20other%20separators%20between%20words.. [Accessed 29 July 2024].