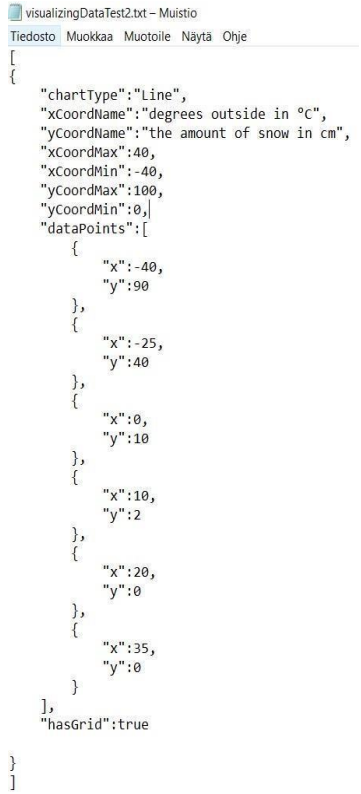
General picture

A programming library that visualizes numeric data. I made a program where the user can input a formatted json-file of their choice and the program will make a line-chart accordingly. The program uses only drawing libraries with individual elements and the graph is only constructed using such tools.

The manual

The program is started from the main-object in the src>main>scala -directory. Then the user sees a screen which asks them to choose a file from their computer to get the data. The file must be a ".txt" -file to be handled correctly by the program and it must be in json-format in order for the execution to go successfully. If the .txt file is not in json format, the user will see exception in the console. If the user chose an incorrect file, they can click the "choose file" -button again and find the correct file. After choosing a file that satisfies the requirements for the expected json file, the data is visualized as a line-chart on the screen.

The json file -input is strictly regulated and should be in the same format as in the example:



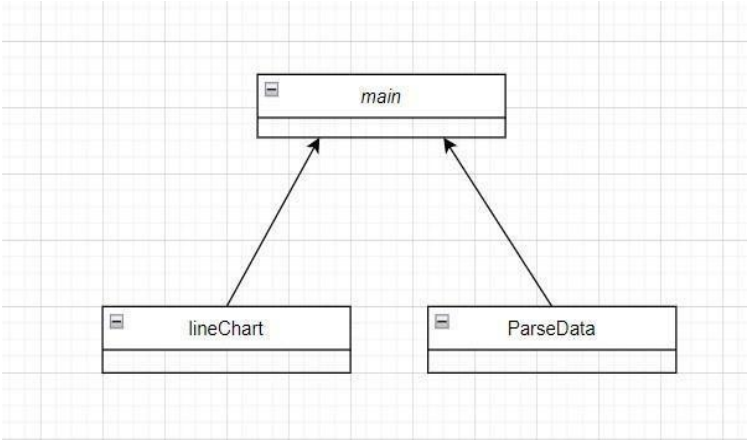
produces:

The chartType input does not matter to the program as there is only one type of chart available. Still it must be there for the program to function correctly. xCoordName and yCoordName are the names of the coordinate axis that the user can choose. The program will handle them as strings. xCoordMax, xCoordMin, yCoordMax and yCoordMin are the max and min values of the range that the user wants the chart to be in. These inputs must be integer -datatypes so that the program does not throw any exceptions. The max inputs must be at least 10 units bigger than the min inputs to avoid errors.

After that, there is a list that contains all the datapoints that the user wants to show called “dataPoints”. The amount of datapoints is not regulated. The datapoints must be within the min and max range of both axis in order to avoid exceptions. Lastly there is the “hasGrid” -input where the user answers either “true” or “false” in accordance with their grid preference.

Program structure

The object in which the program is started is called “main”. There are also two additional classes, “lineChart” and “ParseData”. In main object the whole window for the program is initialized and the most important elements are handled. The main object gets the data from “ParseData” and then gives the data to lineChart class as a buffer. The lineChart class then draws the chart. After that, the lineChart layer is added to the root window in the main object and the user sees the chart.



In main -object there are no remarkable methods, just adding and deleting layers from the window. In the ParseData-class there are readText and readJson -methods that help making the json-file to a nice buffer from which the values are easier to read.

In lineChart-class there is a lot of code for just drawing the charts. Some important methods are the dataPointXInCanvas and dataPointYInCanvas that convert the user input point from the data to certain coordinates on the drawing canvas.

Algorithms

The most important algorithms in the program were the ones needed to convert the user input points to the ones in the drawing window. The y-coordinates had to be taken reversely because for some reason the window coordinates start from top and end at the bottom, which is obviously not the case with my chart values. I found that the formula for x-coordinate is (point-xCoordMin)*(600)/(xCoordMax-xCoordMin) + 200. The odd-looking constants 600 and 200 come from the coordinate points that were the basis of my chart in the canvas.

For y the formula was (yCoordMax - point)*(400)/(yCoordMax-yCoordMin)+300

There were also some algorithms used to draw the strokes that initialized the whole chart and the strokes between each data point, however they were quite simple.

Data types

I used mutable buffers to handle the data and I thought it was a good way to access the data. The data types in the json buffer were different from one another so I had to check that they were correct in other classes that did not know the datatypes beforehand.

Files and network data

The program processes a text (.txt) file and checks if it is in correct json format. There is an example data file that can be used to test the chart in my projects version control. The json format was already described quite in depth in the manual section.

Testing

I mainly tested the program with println console tests where I could see if the buffers had the correct elements or not. I also used a lot of time to experiment with the canvas to draw the coordinates correctly.

The flaws of the program

I am not aware of any flaws in the program at the moment, given that the json is implemented correctly. If the user opens a .txt file that is not in correct json format, then the select file -button disappears and the program must be restarted. If the opened file is not a .txt -file, then a regular exception occurs.

Best and worst spots

I really like how nicely the chart scales if the user changes the max and min coordinates. The chart can even show negative values and it still works fine on both x and y axis. I like the user interface a lot as well. I put a lot effort to think of the perfect spots for each coordinate number strings and line “extensions” from the sides so that it would look pleasant. The only thing I did not like is the error I already talked about where the select file -button disappears once the user selects a wrong text file. Otherwise I am very content with the project

Overall review of the project

I think the outcome was very good. There are no fundamental flaws in the project and I am quite confident that I caught all the errors in the exceptions.

Of course, some things could be improved. One thing I thought of was having better exception messages with the json file -related errors. I could have made the ParseData class better so it would fit other chart inputs as well.

I think that the project would be easy to extend further. I would just need to make another parsedata- and chart-class for each chart and call them in the main object based on the preferred chart type. I also thought

about making a checkbox from which the user could change the state of the Grid. I think that would be quite easy to implement.