PROJECT REVIEW PAPER

by Stanislaw Ryazanov 214-1

Supervised by Julio Cesar Carrasquel Gamez

12.06.2022

PROBLEM POSED.

Specification given asked me to create an application that could parse .csv files, dynamically process user-inputted data, have a graphical display of data and a friendly user interface. Here is the project specification itself:

Table of Contents

Introduction to Programming Project: Stock Portfolio Manager

Project brief:

For the project part of your introduction to Programming Course, you must implement a Stock Portfolio Managing Application. This application will allow the user (a typical investor), to monitor and update information regarding their stock portfolio. Furthermore, the application will provide an interactive UI, for calculating basic portfolio characteristics.

Please find an exemplar data spreadsheet here:

Glossary:

- Company (N)– name of the company.
- Operation (B/S)– buying or selling stock(s).
- Quantity (Q)– Number of stocks bought or sold.
- Current Price (CP) – The price at which the stock was acquired.
- Date – Date at which the stock was acquired.

Formula provided below:

- Net Profits (NP) – Profits earned from stock price increase (can be negative).
- Current Stock Worth (CSW) – The total worth of all stocks in the portfolio.
- Amount of money invested into stocks (AS) – Total amount of money spent on buying stocks.

Features to implement:

Managing Operations:

- Adding a new company to the stock portfolio (N, B/S, Q, CP, Date) - specified by the user).

- The ability to delete stocks from the portfolio (a typical “SELL” operation). You must include: “Sell All”, “Sell by quantity”, “Sell all stocks for one company_name” features.

- Looking at a history of the inputted data. (a spreadsheet inside the application will suffice).

Viewing operations:

- The ability to see a list of companies in which the user owns holdings (this list can be made sortable).

- Viewing detailed information about a particular company (“NP”, “Company CSW” “Company AS “, “Company Q”, “CP”.

- Display of: “CSW”, “AS”, “NP”

- Simple graphical representation about the portfolio.

Useful Formula:

Net Profits:

$$NP = (CS) - (AS)$$

Current Stock Worth (CSW):

$$CSW = \sum_{1}^{n} (Q_i) * (CP_i)$$

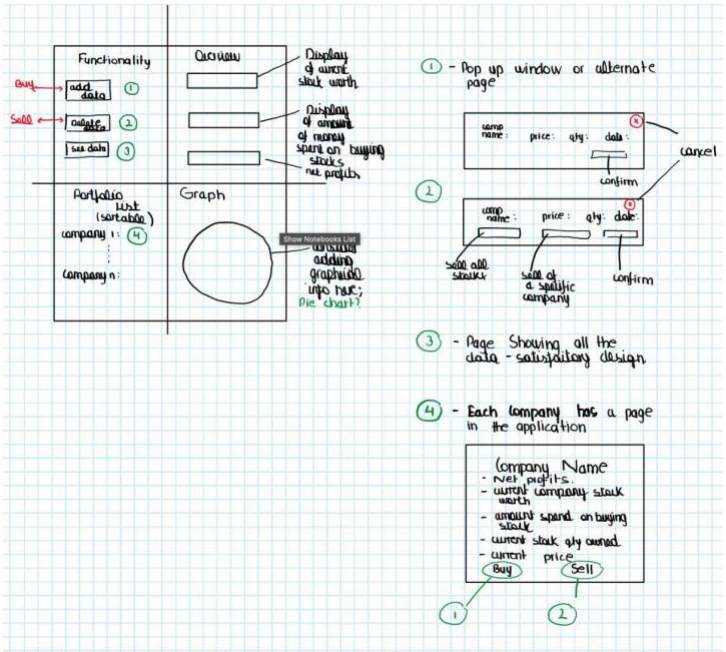
Amount of money invested into stocks (AS):

$$AS = \sum_{1}^{n} \sum_{1}^{L} (Q\ of\ stock\ bought\ at\ a\ price)_{j,i} * (P\ at\ the\ time\ of\ acquistion)_{j,i}$$

*n is the number of companies.
*L is the number of stock acquisitions for a company.

*Respectively, sum from 1 to n can be omitted, when calculating AS & CSW for one company.

Diagram of application requirements:



Advanced features (not mandatory):
-Parse information from a website, so that the current stock price is always visible and updated on project start-up.
-Implement many different graphical representations of the portfolio.
-Make cool interactions.

-Have fun and do whatever you think is best!

IMPLEMENTATION DETAILS.

Project could be accessed via this link.

Firstly, I had to inspect the CSV dataset provided. Here I got a basic idea of how to organize the parsing of each line: I created a struct *Action*, whose purpose was to help the app export the 'final' version of the table including all actions into a separate .csv file. Next, a struct called *Company* was created: it holds all the information about a particular company in it, including name, total worth, total money spent, etc. Then, all of the companies were put into an unordered_map *tableMap*, having their names as keys and the structs as values. It makes it a lot easier to navigate through them by simply accessing a company by its name, and not having to search for it [like in a vector of *Company* structs]. Finally, the dataset was parsed and put into a QTableView widget, which could be easily sorted alphabetically by the names of the companies. After that I implemented three key features: Overview section, Action section and a Pie Chart section.

Overview. Here the user has quick access to various information about their portfolio: Total Worth of their stocks, how much they have spent on them and what are the Net Profits they get. Net Profits are calculated dynamically, meaning that Total Worth for a company is computed via last price used and not just summed up with 'constant' prices.

Actions. A user has to be able to perform a number of actions: buying stocks, selling stocks, and viewing detailed information about a certain company. The latter is the easiest one – a user clicks on a company, whose information they'd like to check, press "Information" and a pop-up window appears, showing detailed information about the stocks of a selected company owned by the user. Buying and Selling stocks functions are similar: an *Action* is created and put into a vector with other actions, then, as the user clicks a button to submit an action, that empty *Action* gets filled with all the necessary information and the data in the unordered_map *tableMap* gets updated (including Total Worth/Spent and Net Profits).

Pie Chart. That was the trickiest part of all: standard QT does not offer such a luxurious option as displaying a simple pie chart, so I had to do it myself using QPainter. I chose Total Worth to be a subject of the chart, as it is quite intuitive (though *rarely* might be tricky, we know cases of having negative prices on certain assets in real life). I needed this Pie Chart to appear in the lower right part of the window, so I used a QLabel and forced a QPixmap into it. In that QPixmap, I drew each of the companies from the unordered_map as a 'pie slice' sized according to a ratio of its stocks worth to Total Worth.

Misc. One of the features I added is selling/buying stocks of a company by clicking in it in the QTableView and pressing "Sell Stocks" or "Buy Stocks" respectively so the user does not have to type the name of the company in. Next neat thing introduced is being able to keep track of user's actions by saving the 'history' of their purchases/sells into a .csv file. And the last feature I've added to my app is the 'About' section, which is a window with my StudentID and a fun motivating logo.

RESULTS.

The final result is a well-operating nice-looking application, that could even be used in real life scenarios. Alongside the development of this application, I have learned to plan the process of creating any piece of my code taking how it would be used in the future into consideration. That includes planning data structures, methods, patterns, etc. Having close to no information on how to create advanced QPainter “images” I have also learned to work with classifications, documentations, and various other online reference sources.

Throughout the process of development of this application I have encountered numerous bugs, both minor and major, but managed to fight all that I could find. Of course, in no way this application is perfect, but I am quite happy with the end result, given it is my first big work of that sort and that it went relatively smooth.