Java Project : Portfolio Manager

Subhan
Hagverdiyev
NHL9KN

Programming 3, Budapest University
of Technology 3
Dr. Goldschmidt Balazs

1 Problem Description

Concept
The concept of this project is to create a programme that allows individuals to create and maintain a stock portfolio composed of shares from a virtual stock market without any financial commitment. This exercise in trading shares over a simulated stock market is intended to provide practice for beginners interested in stock markets.

Goal
The goal of this project is to create and implement a stock trading game, which successfully simulates a simple portfolio manager and is capable of the following functions:

- Displaying a stock market that continuously updates its display data (so-called Christmas tree application)
- Opening and maintaining a portfolio that accurately tracks changes in the market
- Building a portfolio from a choice of different shares
- Investing in shares
- Selling shares
- Increasing / decreasing available balance
- Calculating profit/loss

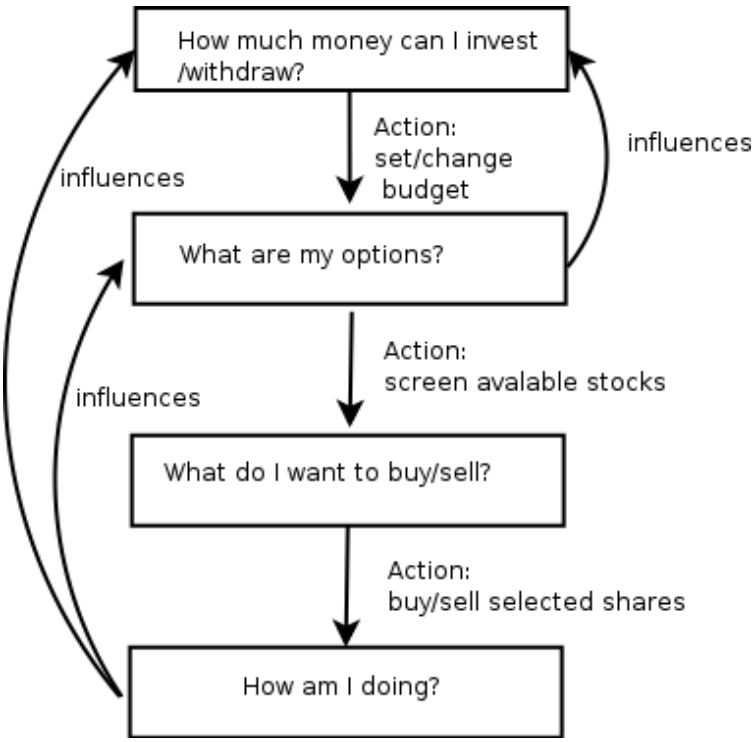
Moreover, a comprehensive graphical user interface (GUI) is intended to provide optimal display of data while also simplifying communication between the user and the software.

Proposed Solution
In order to formulate an appropriate solution that would realise the above concept, it is vital to consider the time frame within which the project is to be finished, which in this case is approximately 8 weeks.

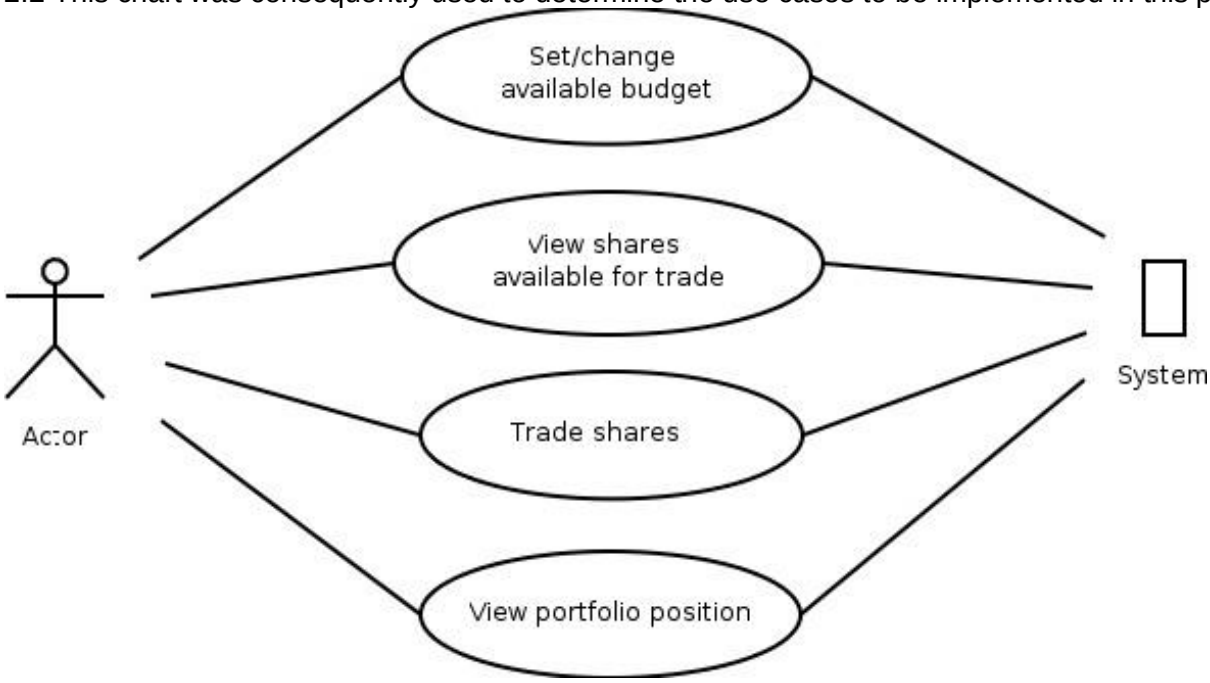
The project is created and implemented in Java along guidelines provided by Object Oriented Programming (OOP).

2 Problem Analysis, Design and Implementation

2.1 In order to understand the specific requirements of such a programme, a simple flow-chart demonstrating the thought process of a user was created:



1. 2.2 This chart was consequently used to determine the use cases to be implemented in this project:



2.

2.2 Use Case Descriptions

Use Case Name	**Set/Change available budget**
Person in charge	
Priority	

Pre-Conditions	•JRE has been installed to run programme •Programme is activated •A new portfolio is being or has been created
Post-Conditions	• **Portfolio frame is open** • **Stock market display is open**
Trigger	• User clicks the button 'Sign up, 'Sign In' 'Add balance' or 'Withdraw Balance'
Non-functional Boundary conditions	
Normal Flow	Open programme > create new portfolio > specify balance to start with > open portfolio > add or withdraw balance by clicking the respective buttons
Error	• User does not supply a number variable as amount • User wishes to withdraw more than his account balance
Result	• Balance updated accordingly OR error message displayed
Notes	NOTE: Negative balance not permitted
Unclarified points	

Use Case Name	**View shares available for trade**
Person in charge	
Priority	

Pre-Conditions	•JRE has been installed to run programme •Programme is activated •Stock market frame is open
Post-Conditions	• **Stock market frame is open**
Trigger	• User opens application
Non-functional **Boundary conditions**	
Normal Flow	Open application > stock market display with a list of all available shares is opened > User maximises/minimises window
Error	• Application is not started properly • Internal error in obtaining display data
Result	• **All shares available for trade are displayed in a table along with their prices and deltas**
Notes	
Unclarified points	

Use Case Name	**Buy shares (Trade shares)**
Person in charge	
Priority	

Pre-Conditions	•JRE has been installed to run programme •Programme is activated •A new portfolio has been created •Positive portfolio balance
Post-Conditions	• **Application is open** • **Trade does not reduce portfolio balance below zero**
Trigger	•User clicks 'Buy' button on portfolio frame
Non-functional **Boundary conditions**	
Normal Flow	Open application > create new portfolio > click the 'Buy' button > a list dialog with all shares available for purchase is shown > choose share > specify quantity (number) > click SET button > portfolio table appended to display purchase > displayed portfolio details updated to reflect purchase
Error	• User does not feed in an integer variable as quantity • Portfolio balance is zero • Trade would reduce portfolio balance below zero
Result	• Portfolio table displays purchase made • Displayed portfolio details updated to reflect purchase • OR error message displayed
Notes	NOTE: Negative balance not permitted
Unclarified points	

Use Case Name	Sell shares (Trade shares)
Person in charge	
Priority	

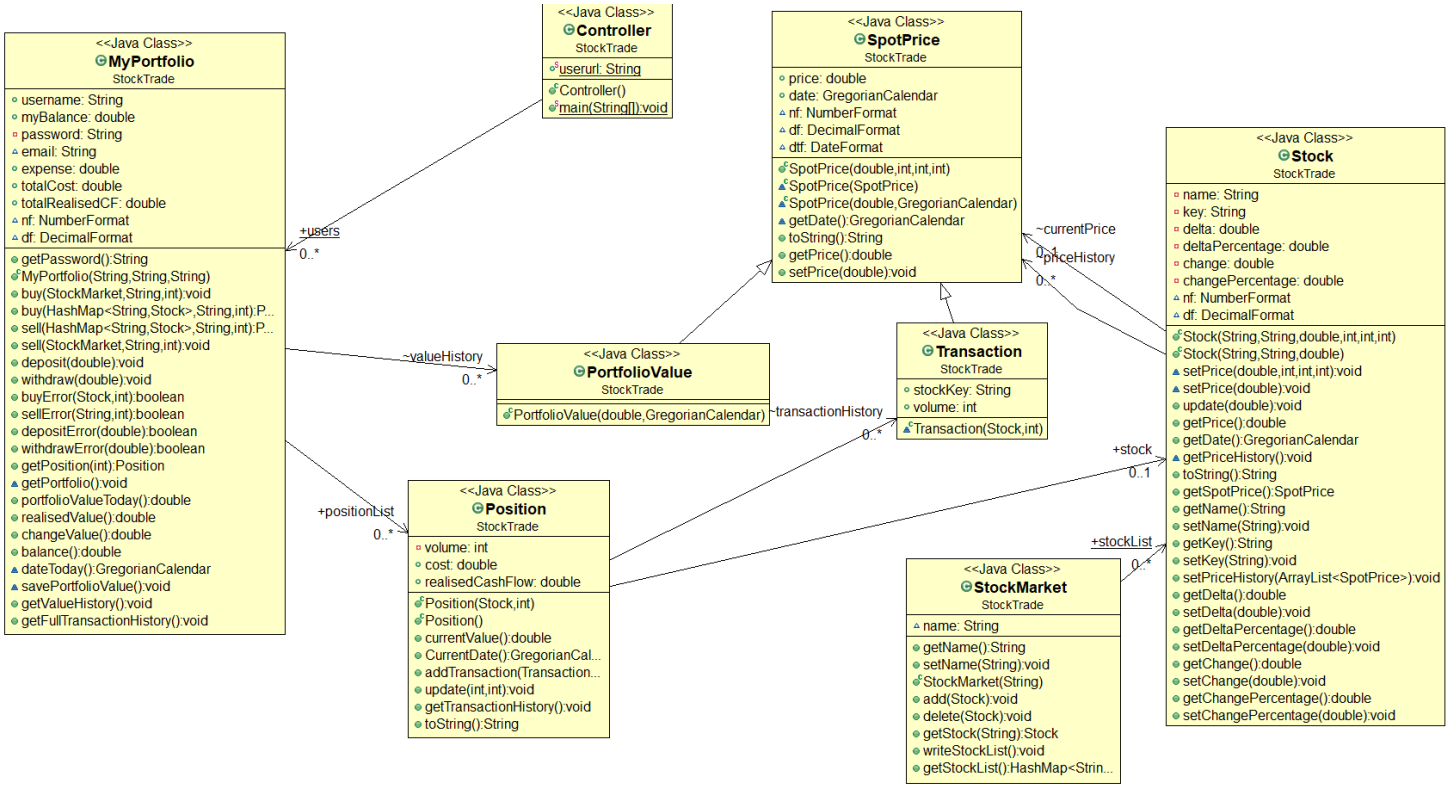
Pre-Conditions	•JRE has been installed to run programme •Programme is activated •A new portfolio has been created •Portfolio contains one or more of the share to be sold
Post-Conditions	• **Application is open** • **Trade does not reduce quantity of share held in portfolio to below zero**
Trigger	•User clicks 'Sell' button on portfolio frame
Non-functional Boundary conditions	
Normal Flow	Open application > create new portfolio > add shares to portfolio > click the 'Sell' button > a list dialog with all shares available for sale is shown > choose share > specify quantity (number) > click SET button > portfolio table appended to display sale > displayed portfolio details updated to reflect sale
Error	• User does not feed in an integer variable as quantity • Quantity of share wished to be sold equals or is less than zero • Trade would reduce quantity of share held in portfolio to below zero
Result	• Portfolio table displays sale made • Displayed portfolio details updated to reflect sale • OR error message displayed
Notes	NOTE: Short-selling not permitted
Unclarified points	

Use Case Name	View portfolio position
Person in charge	
Priority	

Pre-Conditions	•JRE has been installed to run programme •Programme is activated •A new portfolio has been created
Post-Conditions	• **Application is open**
Trigger	•User opens a new portfolio (Clicks 'Create Portfolio)
Non-functional Boundary conditions	
Normal Flow	Open application > create new portfolio with starting balance > (add/ withdraw balance OR change portfolio composition OR stock prices change) > displayed portfolio position details updated to reflect changes
Error	• Portfolio frame not opened according to instructions
Result	• Opened portfolio frame displays portfolio position details
Notes	
Unclarified points	

2. 2.2 Class Diagrams and Descriptions

- Package StockTrade:



Class Descriptions for Package StockTrade

This package contains the following classes that implement the market and portfolio structure behind the scenes of the GUI:

- Controller : This is the controller of the program.It has a list of MyPortfolio class objects to save the users in list. Program starts from here thus contains the main function and it calls the SignUp class by starting application.NOTE: I have written my own path to userurl variable if your path is different you can simply change variable's value and it will change in everywhere.

- StockMarket(String name): An instance of this class is a stock market where all the stocks are stored in a map. It has the following functions that are critical (used by GUI to display a table of all available stocks):

  void add(Stock stock): adds stock to stockList
  -void delete(Stock stock): deletes stock from stockList
  -Stock getStock(String stockKey): searches and returns a stock from stockList
  -HashMap getStockList(): returns stockList

- Stock(String key, String name, double price): An instance of the stock class represents a single stock identified by a unique stock key, which is then stored in a stock market. This class stores all information concerning a particular stock (price history, which is a list of spot prices over time, name, key, delta). The following functions are of particular importance:

  -double getPrice(): returns the current stock price
  -void setPrice(double price): sets the price for the current date while adding the old price to priceHistory
  -void update(double delta): an alternative to setPrice(), this method changes the current price by a given delta
  -SpotPrice getSpotPrice(): returns the current SpotPrice for further calculations
  -GregorianCalendar getDate(): returns the current date

- SpotPrice(double price, Date date): This class can only be instantiated by a stock. An instance of this class has a date and a price and is dependant on a stock for its existence. It has three constructors to allow flexibility. Any price change undertaken by the Stock class changes the price in a certain spot price (at a given date). Its getPrice() and getDate() methods are used by the corresponding methods in the class Stock.

- MyPortfolio(String username, String email, String password) : This is the heart of the application and contains all functions that allow the user to buy and sell shares, to deposit or withdraw money and to calculate the current value and cost of a user portfolio. An instance of this class has a map of positions (in different stocks) identified by the stock key. A purchase or sale of shares would cause the composition of this map to change. However, positions are never completely eliminated from this map. Even after all units of a stock have been sold, the position persists with a quantity value of zero. The portfolio also contains a map of all its current values (PortfolioValue), which could then be used to create a time series graph of the development of its value. Some of its most important methods are explained:

  - Position buy(HashMap stockList, String key, int qty): buys the given qty of the share for the portfolio and returns the resulting position

-Position sell(HashMap stockList, String key, int qty): sells the given qty of the share for the portfolio and returns the resulting position
-void deposit(double bal): increases portfolio balance by the given amount
-void withdraw(double bal): decreases portfolio balance by the given amount
-void loaduser(): this method is used for Java serialization for loading users.
-void saveUser(): this method is used for Java serializttion for saving users.

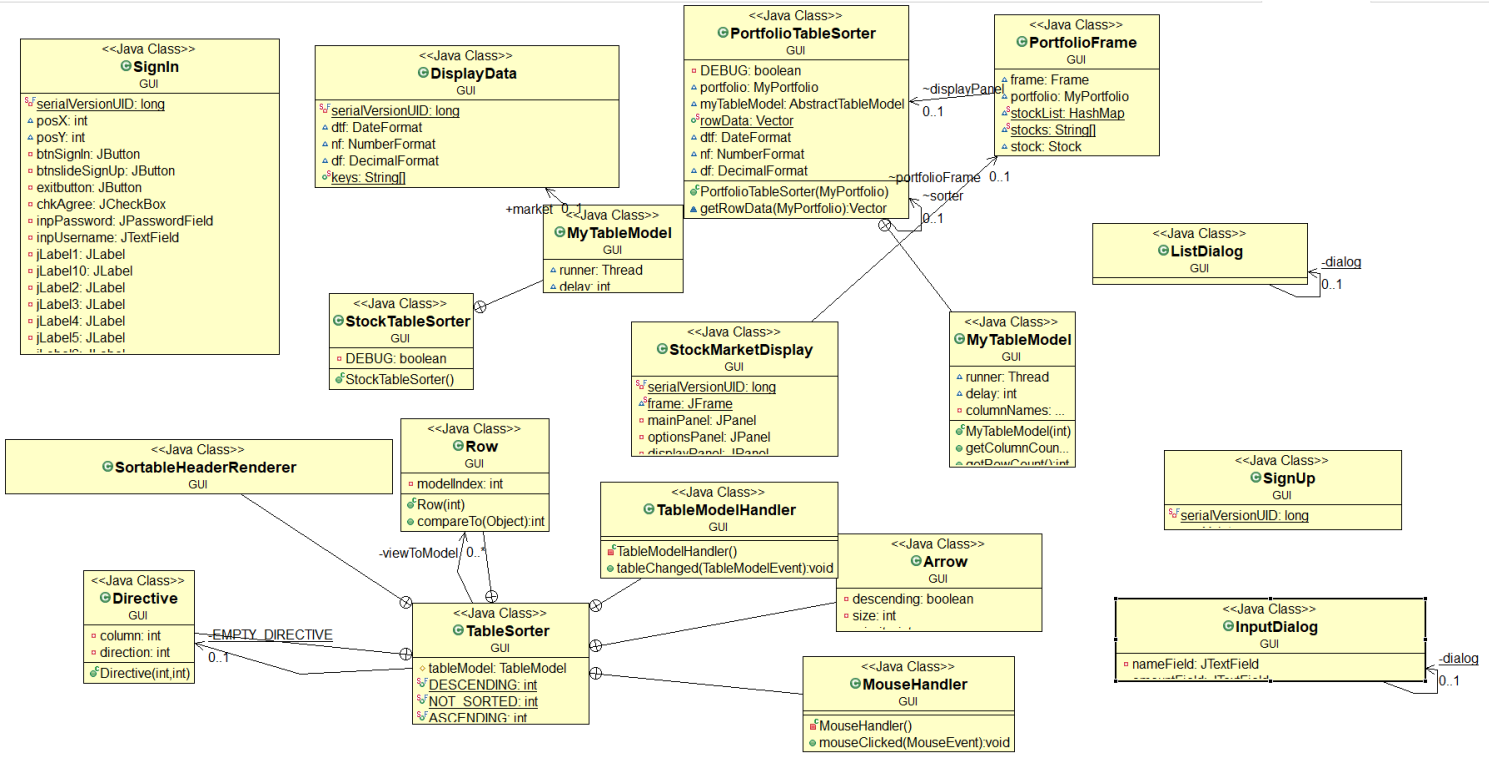
- Position: This class can only be instantiated by a portfolio and depends on the portfolio for its existence. It is important to note that one position can only point to one stock (and vice versa). Hence, when new units of an already purchased stock are bought, the respective position is updated instead of creating a new position. A position has a list of transactions, which stores all transactions undertaken for the particular stock it represents. Any current value/ cost calculations undertaken by the portfolio aggregates values that are calculated in its positions. It has following functions that are of importance:

  -double currentValue(): returns the current value of a position
  -void update(int qty, int id): updates the cost of a position and is used by the buy and sell methods of MyPortfolio
  - void addTransaction(Transaction transaction): adds a transaction to its transaction list, which is then used by the update method.

- PortfolioValue: A subclass of SpotPrice, it has similar functions. It stores the current portfolio value for a given date. It has no other functions and serves solely to be stored in a list, which charts the development of the portfolio value.

- Transaction: Also a subclass of SpotPrice, a transaction stores the stock key, quantity bought/sold of the particular stock, and the spot price at the moment of sale/purchase. It is vital to the whole structure since any reduction in portfolio cost requires this information in order to be implemented.

  Package GUIStockTrade



Please note that the complexity of the individual classes prevents all classes from being displayed in detail. Moreover, the classes serve only to implement the methods contained in the StockTrade package (see above). GUIStockTrade imports this package in order to access its functions.

A detailed view of the most important classes in the GUIStockTrade is provided on the next page.

Class Descriptions for Package GUIStockTrade:

This package contains the following classes that implement the user interface:

- SignUp: This class is for sign up page where user can sign up or swith to sign in option. The class extends Jframe and implements ActionListener for buttons. After signin up it calls Myportfolio.saveUser() method also add the users to the list which I have created in Controller class to keep track of Users list.
- SignIn: This class is for signing in to your portfolio profile. We can also switch to sign up option with the button top of the page. For checking user's data first we load the users data from files then checking the user's list one by one to match. If the credentials is incorrect the Joptionpane window is opened and says enter the correct details.

- DisplayData: This class implements the runnable interface. It instantiates the required stock markets and provides the data for the stock market table (StockTableSorter). It also implements the thread that generates random price changes observed in the stocks. In addition, this class contains various helper methods such as one that provides a fancy font (provided by Sun Microsystems), one that returns a sorted array of keys for a given HashMap etc. This class is instantiated only once while creating the stock market table. Most of the methods and variables in this class are static.

- StockMarketDisplay: This is the fwindow to confront the user when the application is run and users logged in. It instantiates all the components (containers, buttons, dialogs) that make up the stock market interface and displays the stock market table. It also creates a new portfolio window when the appropriate action is taken. Both the stock market and the portfolio are created as internal frames nested in a desktop frame.

- PortfolioFrame: PortfolioFrame is a subclass of JInternalFrame. This is the heart of the GUI and is the second window to be created (when user creates a portfolio). Accordingly, it is more involved than the StockMarketDisplay. It implements all functions required of the portfolio and creates all the associated components. It supplies the information required to create the portfolio table (PortfolioTableSorter) and displays it. In addition, it displays the current status of the portfolio, allows the user to buy and sell stocks, to deposit or withdraw money and to obtain stock market tips.

- StockTableSorter: This is actually a subclass of JPanel that contains a table using a custom table model. The table model implements the runnable interface and obtains its data from DisplayData (this is the only class that instantiates DisplayData). It runs the thread that publishes changes to stock prices generated by display data.

- PortfolioTableSorter: Almost identical to the StockTableSorter with the only exception that its data is provided by the portfolio frame (i.e., by the user). It implements a thread that reflects the changes taking place in the stock table in the portfolio table.

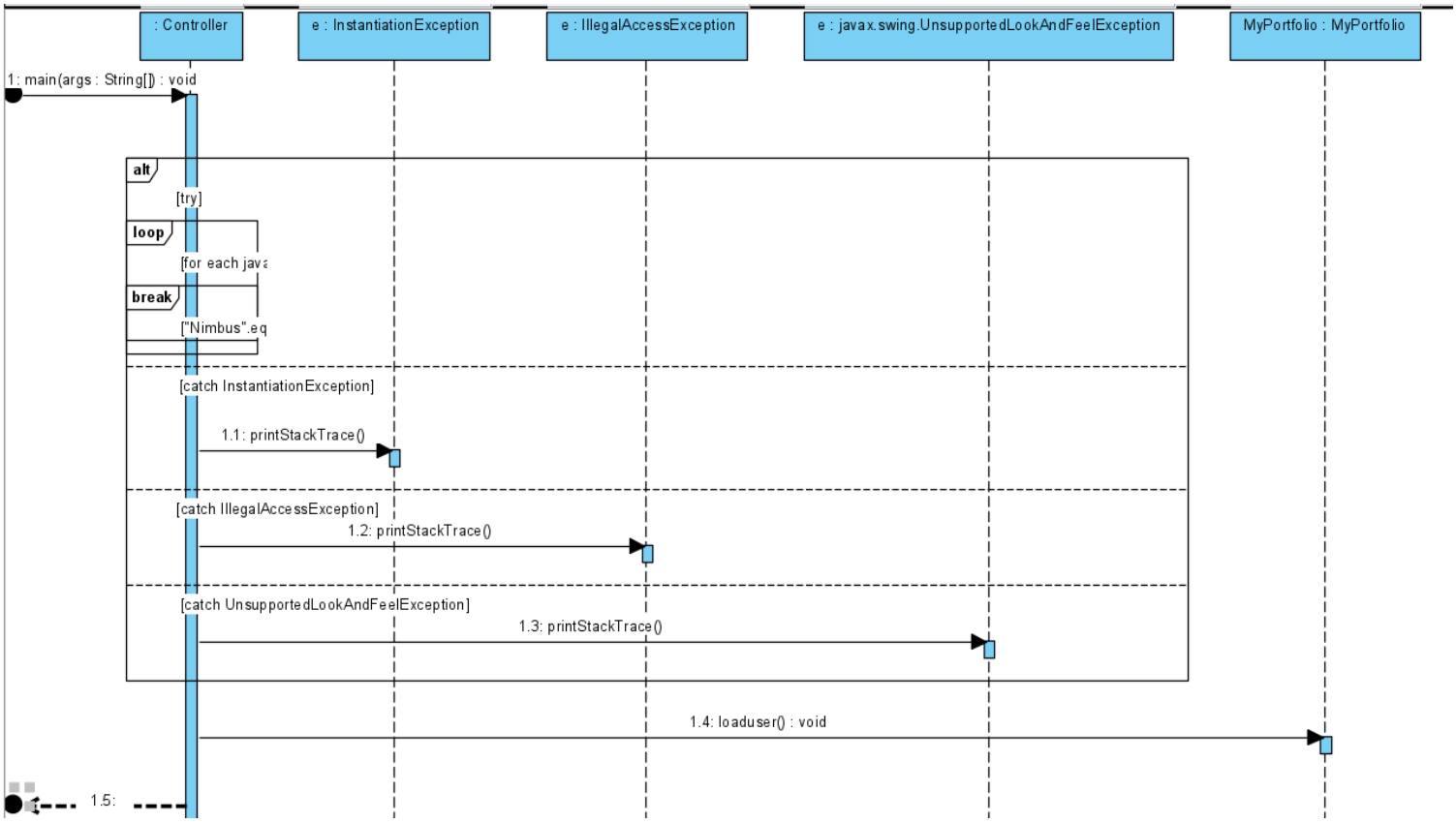
- TableSorter: This class is a subclass of the AbstractTableModel and is provided by Sun Microsystems. It has the sole purpose of decoration and is used to sort the stock market and portfolio tables. This is done by inserting a table sorter between a table and its table model. This application uses the table sorter to allow the user to sort the stock market and portfolio tables by any chosen column parameter. In the beginning of class I have written the name of author and reference for this code.

- InputDialog: This class allows two types of input dialogues to be created (with one or two text fields) according to the specified requirements. Its only function is to deliver data supplied by the user. Both stock market and portfolio windows use it.

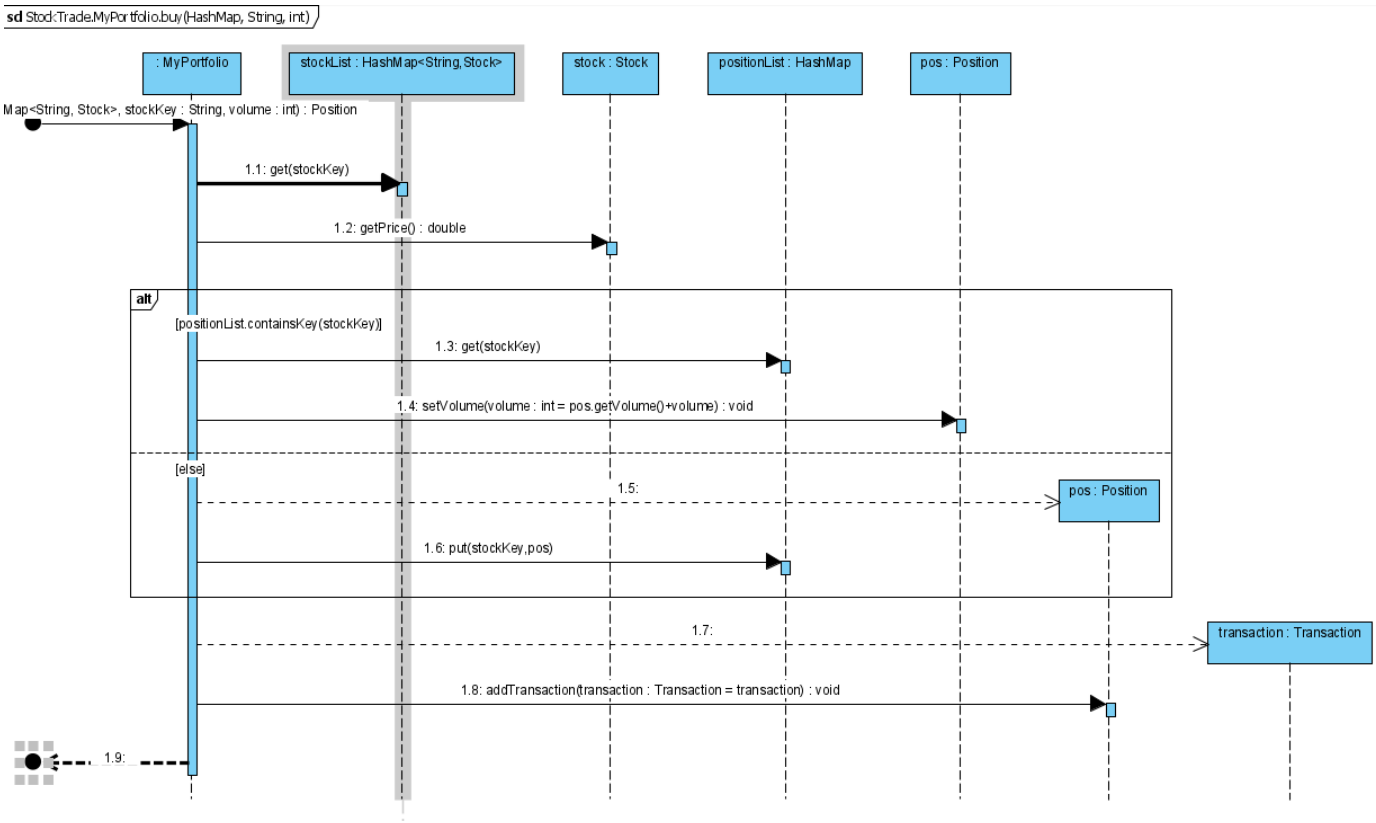
- ListDialog: Almost identical to InputDialog with the exception that it has one text field and one list from which the user is to choose an appropriate option. It is used by the portfolio frame to implement its ‘buy’ and ‘sell’ methods.

2.3 Sequence Diagrams (for the methods ‘buy’ and ‘sell’ from the package StockTrade; for a detailed view of both the diagrams see files ‘SequenceDiagBuy’ and ‘SequenceDiagSell’)

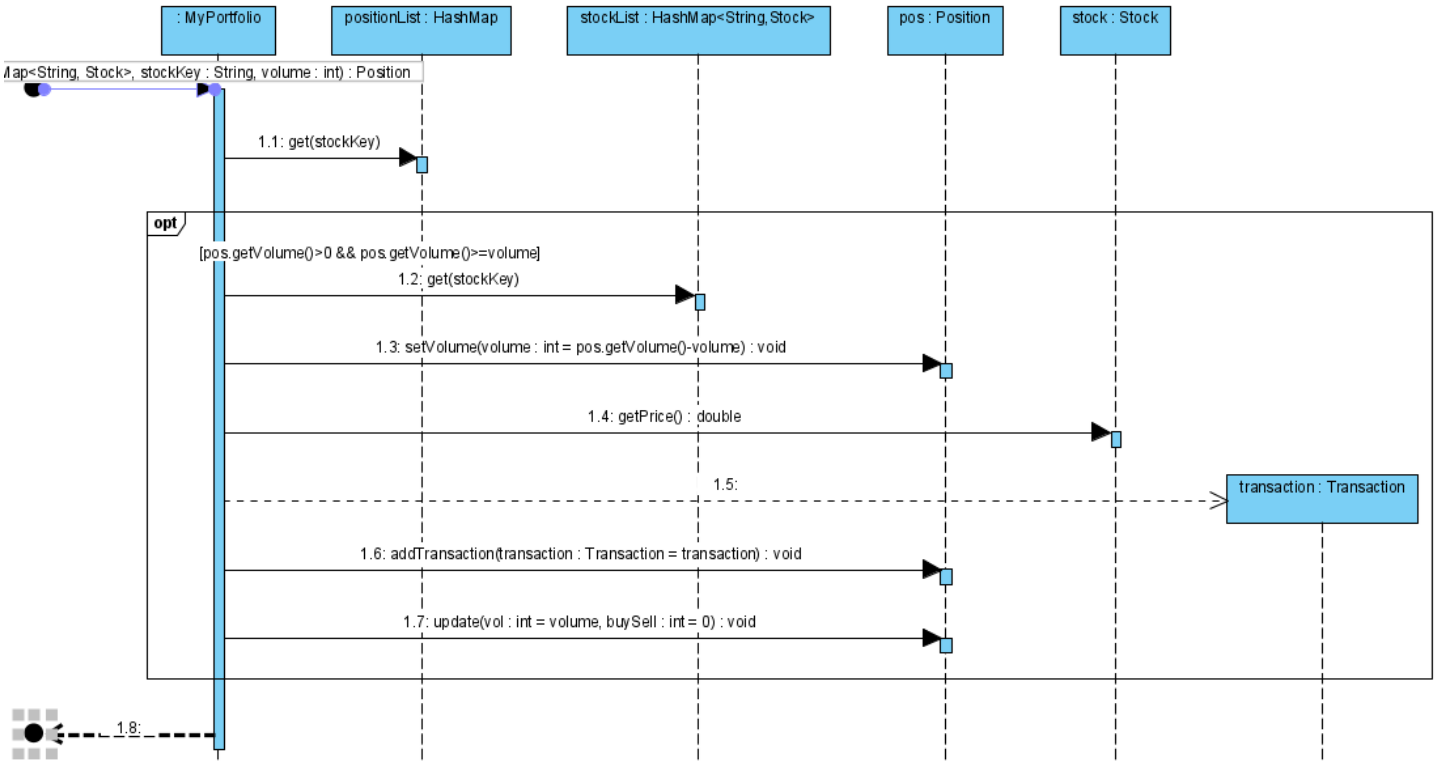
- void main(`String[] args`)



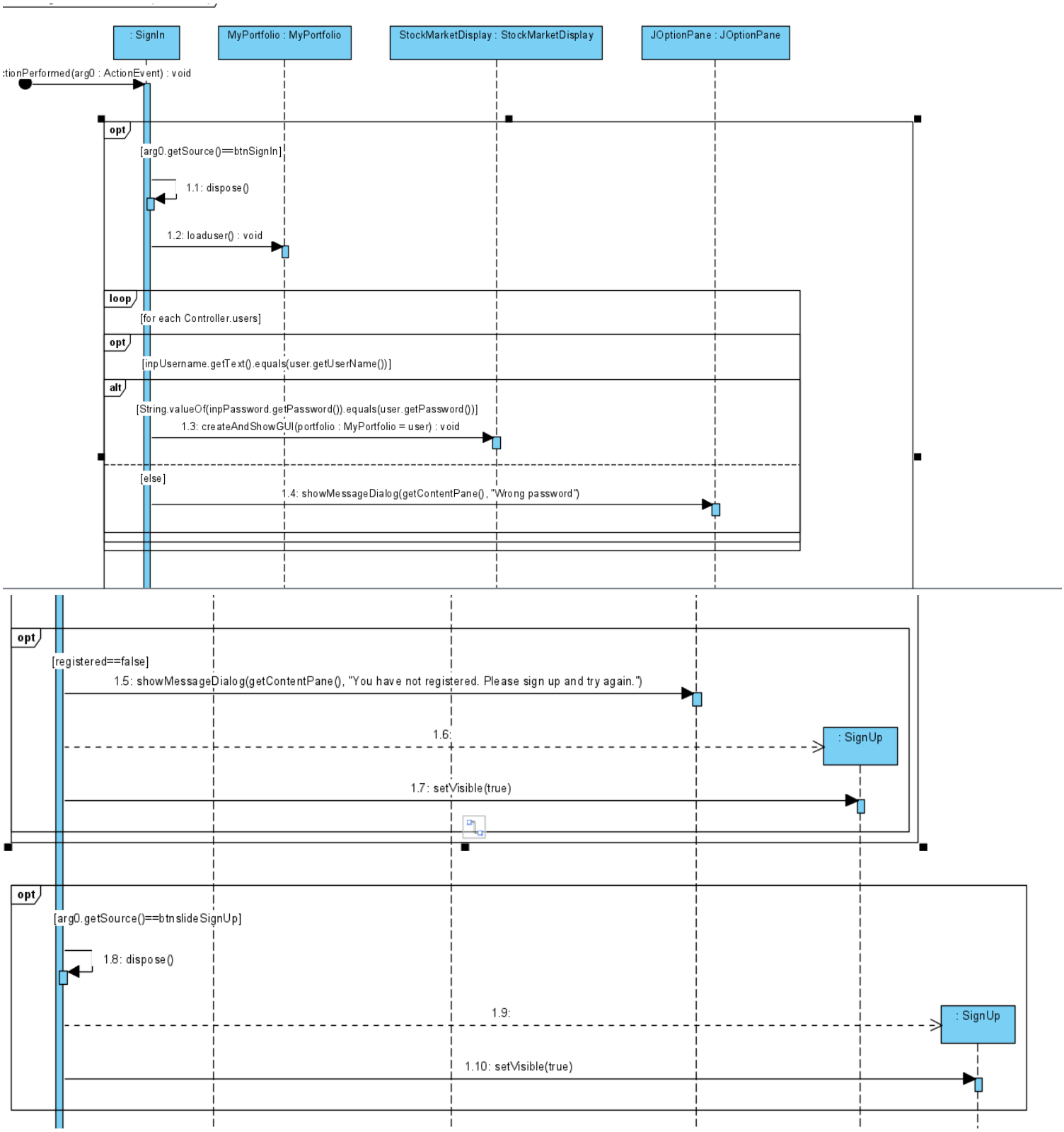
- void buy(HashMap stockList, String stockKey, int qty)



- void sell(HashMap stockList, String stockKey, int qty)

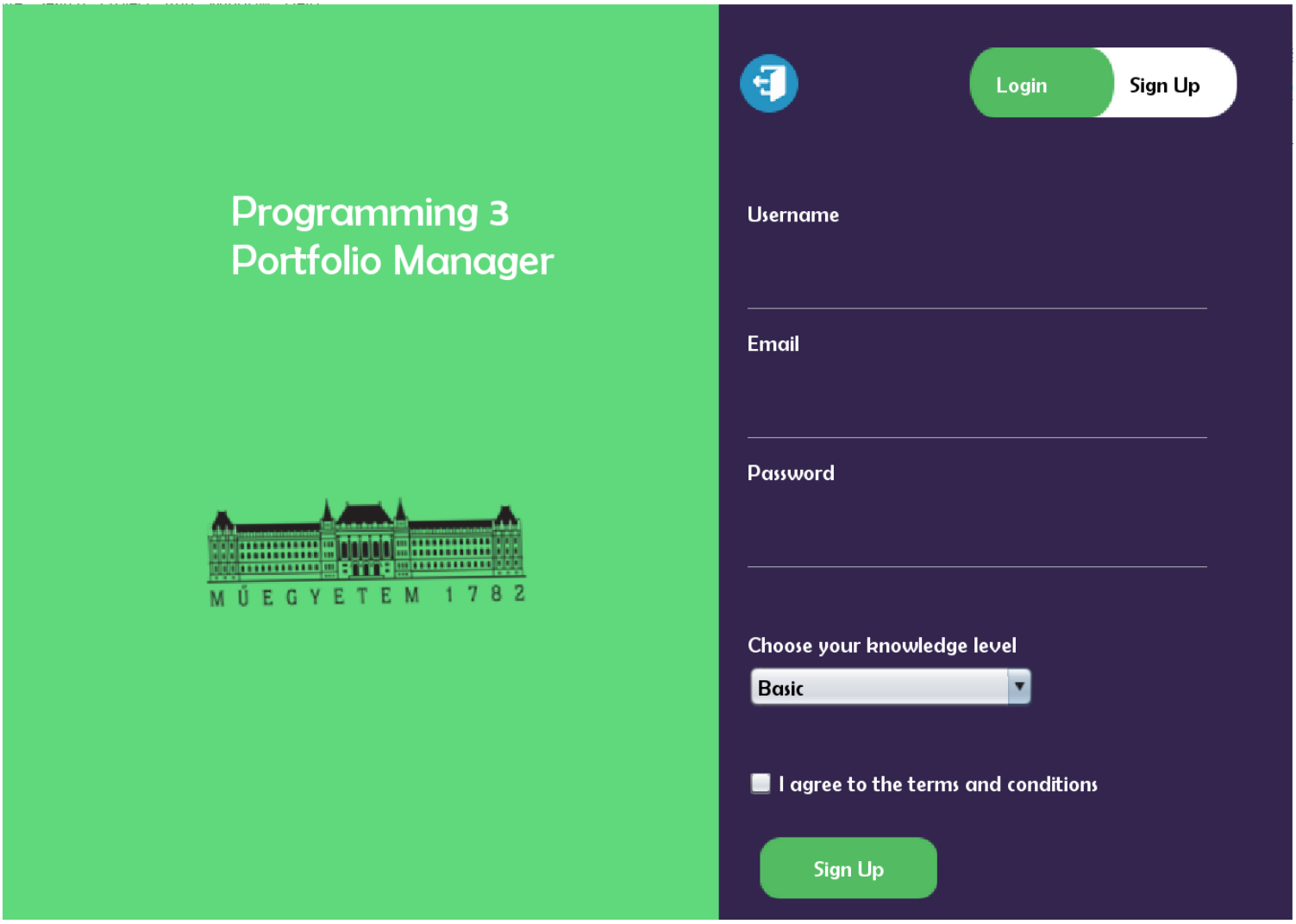


- void SignIn.actionPerformed()

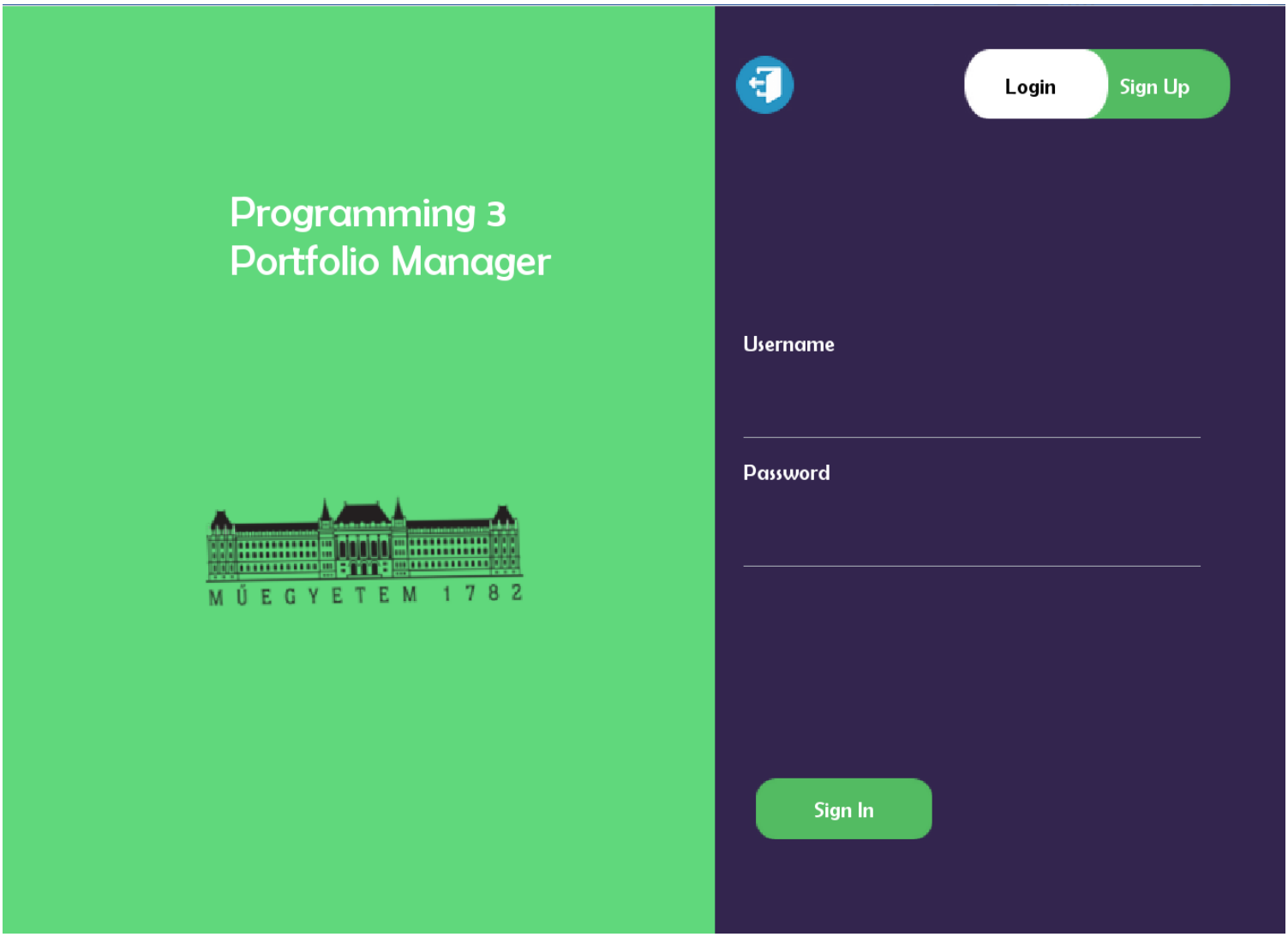


3 Use

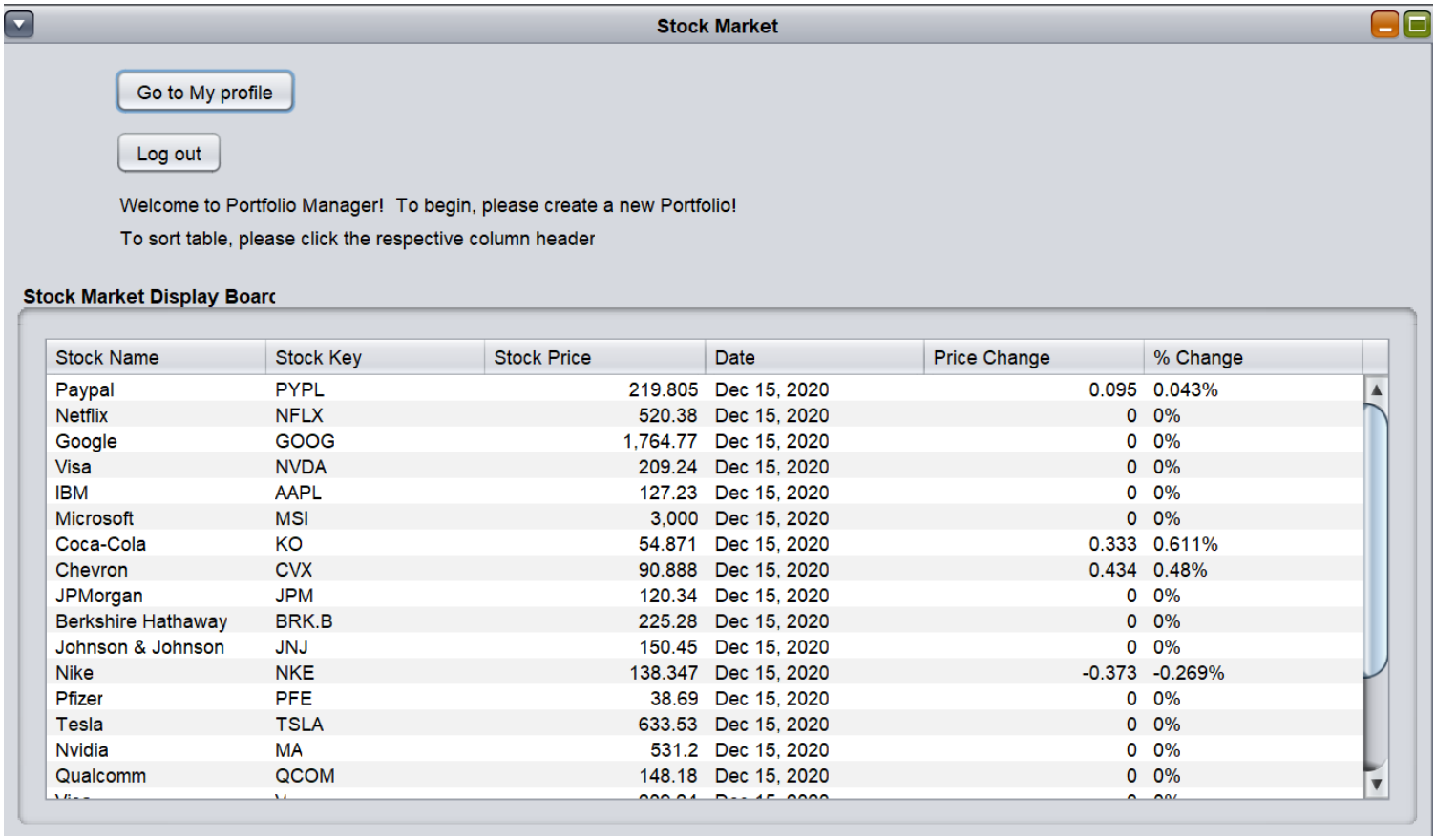
Signup page


Programming 3
Portfolio Manager
MŰEGYETEM 1782
Login
Sign Up
Username
Email
Password
Choose your knowledge level
Basic
I agree to the terms and conditions
Sign Up

Login page:

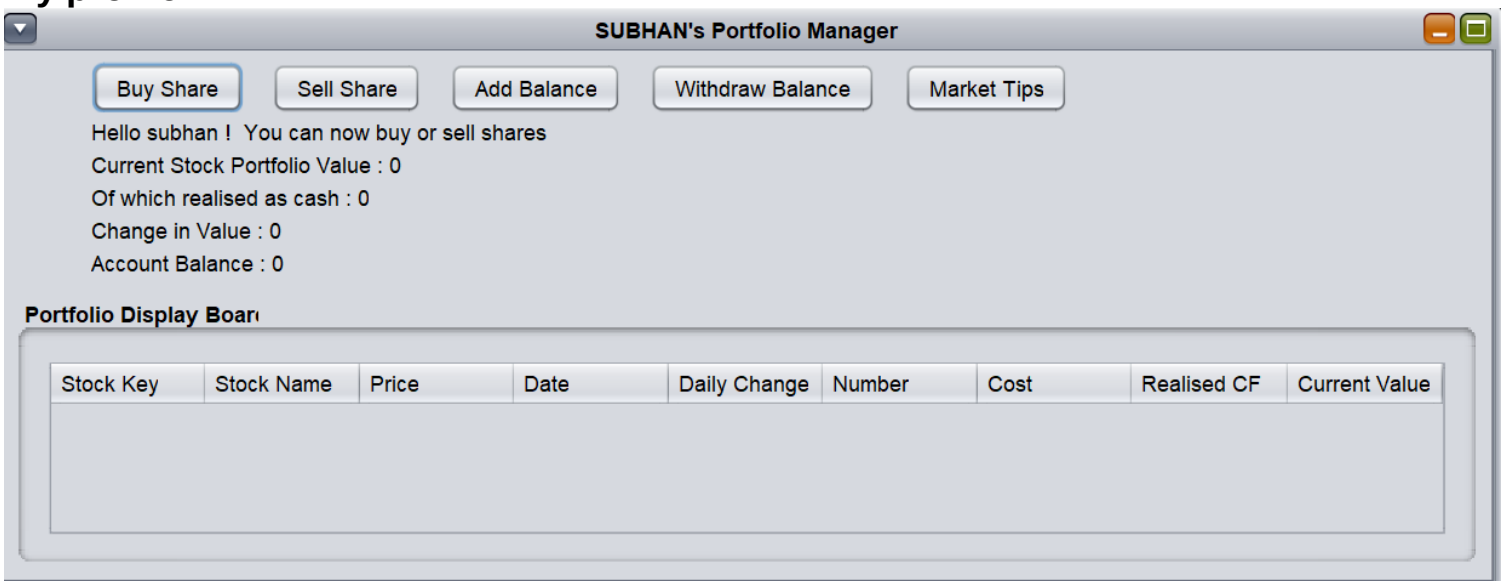


After Logging in:



Stock Name	Stock Key	Stock Price	Date	Price Change	% Change
Paypal	PYPL	219.805	Dec 15, 2020	0.095	0.043%
Netflix	NFLX	520.38	Dec 15, 2020	0	0%
Google	GOOG	1,764.77	Dec 15, 2020	0	0%
Visa	NVDA	209.24	Dec 15, 2020	0	0%
IBM	AAPL	127.23	Dec 15, 2020	0	0%
Microsoft	MSI	3,000	Dec 15, 2020	0	0%
Coca-Cola	KO	54.871	Dec 15, 2020	0.333	0.611%
Chevron	CVX	90.888	Dec 15, 2020	0.434	0.48%
JPMorgan	JPM	120.34	Dec 15, 2020	0	0%
Berkshire Hathaway	BRK.B	225.28	Dec 15, 2020	0	0%
Johnson & Johnson	JNJ	150.45	Dec 15, 2020	0	0%
Nike	NKE	138.347	Dec 15, 2020	-0.373	-0.269%
Pfizer	PFE	38.69	Dec 15, 2020	0	0%
Tesla	TSLA	633.53	Dec 15, 2020	0	0%
Nvidia	MA	531.2	Dec 15, 2020	0	0%
Qualcomm	QCOM	148.18	Dec 15, 2020	0	0%

My profile:



Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value

Maintenance

The application as it currently exists requires no explicit maintenance since changes in prices are randomly generated and it's a single period model for a single user.

However, should this application be developed into a full-fledged portfolio manager, incorporating a larger number of stocks and real-time share data, maintenance would become an integral part of the application. Following points are to note while further developing this model:

- Due to the large number of collections and hence the large volume of data in this application, it is important to efficiently organise the data required to be displayed and updated in the tables. The current model does not allow for large volumes of data to be updated quickly. An alternative would be to assign hash codes to each row data so that specific rows can be updated without scanning the whole list.

- It is important to recognise where the data is coming from. If it is being sourced from an internet resource (such as the web site of a stock market), data needs to be checked on a regular basis.

- Additional functions such as multiple stock markets and multiple users would accordingly multiply the maintenance required.

3 Conclusion

Initially, the project was conceived to process real-time share data, provide graphs and implement a Serializable interface in order to save personal settings. However, the project was scaled down to reflect the realistic time commitment possible (the time to completion needed to be completed in approximately 96 hours, assuming 12 productive hours a weeks) and prevented many of the more sophisticated functionalities from being implemented. In spite of the down-scaling, the time to completion was underestimated