

# Java Project : Portfolio Manager

Shuba Narasimhan  
h0352854

Java Programming Practical, WS 2005/2006  
Business Information Technology  
Dr. Michael Hahsler

## Contents

	<u>Page</u>
1 Problem Description.....	2

2	Problem Analysis, Design and Implementation.....	3
2.1	Analysis Flow Chart.....	3
2.2	UML Use cases.....	3
2.3	Class Diagrams and Descriptions.....	9
2.4	Sequence Diagrams.....	16
3	Installation and Use.....	18
4	Maintenance.....	29
5	Conclusion.....	30
6	Bibliography.....	31

## 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

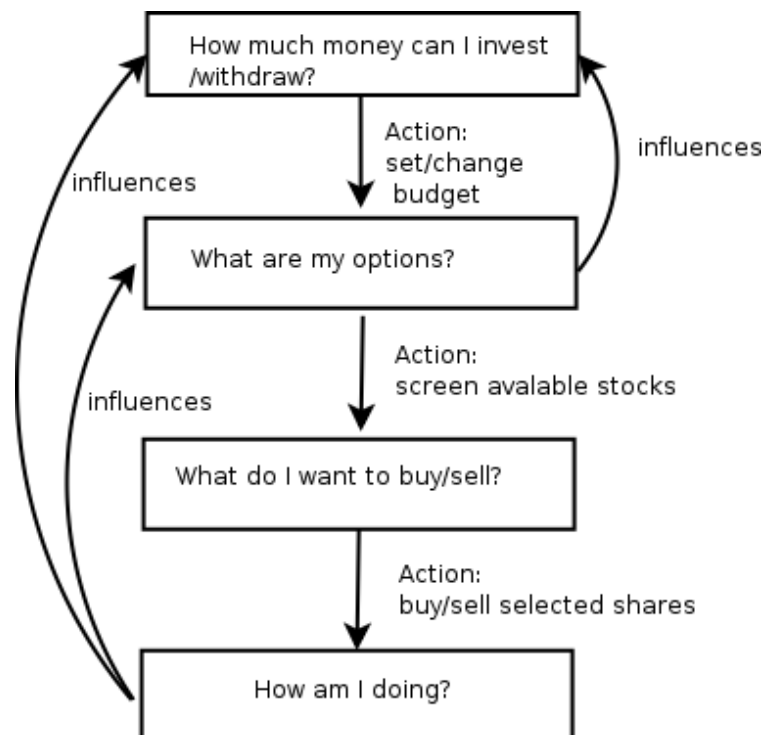
In order to formulate an appropriate solution that would realise the above concept, it is vital to consider the time frame within which the project is to be finished, which in this case is approximately 12 weeks.

Hence, for the sake of maintaining simplicity, the programme is to be initially implemented as a single period model with the possibility of only one user. This could later be extended to accommodate additional functionalities.

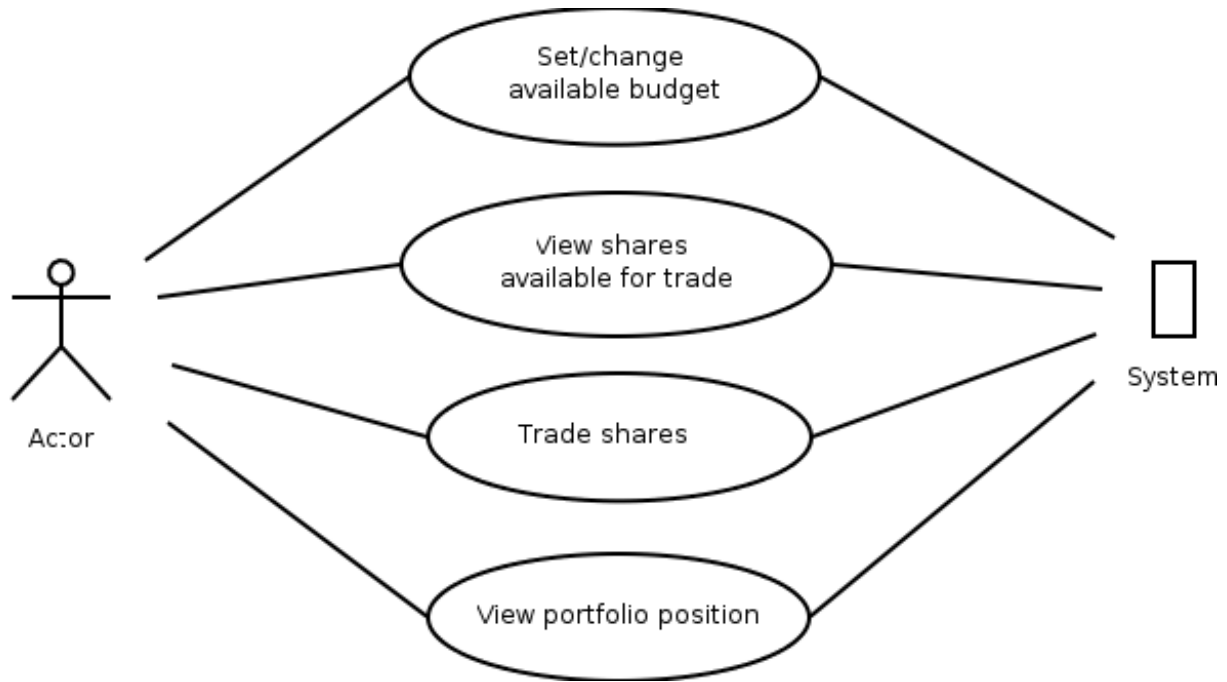
The project is created and implemented in Java along guidelines provided by Object Oriented Programming (OOP).

## 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:



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



## 2.2 Use Case Descriptions

<b>Use Case Name</b>	<b>Set/Change available budget</b>
<b>Person in charge</b>	
<b>Priority</b>	

<b>Pre-Conditions</b>	<ul style="list-style-type: none"> <li>•JRE has been installed to run programme</li> <li>•Programme is activated</li> <li>•A new portfolio is being or has been created</li> </ul>
<b>Post-Conditions</b>	<ul style="list-style-type: none"> <li>• <b>Portfolio frame is open</b></li> <li>• <b>Stock market display is open</b></li> </ul>
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the button 'Create Portfolio', 'Add Balance' or 'Withdraw Balance'</li> </ul>
<b>Non-functional Boundary conditions</b>	
<b>Normal Flow</b>	Open programme > create new portfolio > specify balance to start with > open portfolio > add or withdraw balance by clicking the respective buttons
<b>Error</b>	<ul style="list-style-type: none"> <li>• User does not supply a number variable as amount</li> <li>• User wishes to withdraw more than his account balance</li> </ul>
<b>Result</b>	<ul style="list-style-type: none"> <li>• Balance updated accordingly OR error message displayed</li> </ul>
<b>Notes</b>	NOTE: Negative balance not permitted
<b>Unclarified points</b>	

<b>Use Case Name</b>	<b>View shares available for trade</b>
<b>Person in charge</b>	
<b>Priority</b>	

<b>Pre-Conditions</b>	<ul style="list-style-type: none"> <li>•JRE has been installed to run programme</li> <li>•Programme is activated</li> <li>•Stock market frame is open</li> </ul>
<b>Post-Conditions</b>	<ul style="list-style-type: none"> <li>• <b>Stock market frame is open</b></li> </ul>
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User opens application</li> </ul>
<b>Non-functional Boundary conditions</b>	
<b>Normal Flow</b>	Open application > stock market display with a list of all available shares is opened > User maximises/minimises window
<b>Error</b>	<ul style="list-style-type: none"> <li>• Application is not started properly</li> <li>• Internal error in obtaining display data</li> </ul>
<b>Result</b>	<ul style="list-style-type: none"> <li>• <b>All shares available for trade are displayed in a table along with their prices and deltas</b></li> </ul>
<b>Notes</b>	
<b>Unclarified points</b>	

<b>Use Case Name</b>	<b>Buy shares (Trade shares)</b>
<b>Person in charge</b>	
<b>Priority</b>	

<b>Pre-Conditions</b>	<ul style="list-style-type: none"> <li>•JRE has been installed to run programme</li> <li>•Programme is activated</li> <li>•A new portfolio has been created</li> <li>•Positive portfolio balance</li> </ul>
<b>Post-Conditions</b>	<ul style="list-style-type: none"> <li>• <b>Application is open</b></li> <li>• <b>Trade does not reduce portfolio balance below zero</b></li> </ul>
<b>Trigger</b>	<ul style="list-style-type: none"> <li>•User clicks 'Buy' button on portfolio frame</li> </ul>
<b>Non-functional Boundary conditions</b>	
<b>Normal Flow</b>	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
<b>Error</b>	<ul style="list-style-type: none"> <li>• User does not feed in an integer variable as quantity</li> <li>• Portfolio balance is zero</li> <li>• Trade would reduce portfolio balance below zero</li> </ul>
<b>Result</b>	<ul style="list-style-type: none"> <li>• Portfolio table displays purchase made</li> <li>• Displayed portfolio details updated to reflect purchase</li> <li>• OR error message displayed</li> </ul>
<b>Notes</b>	NOTE: Negative balance not permitted
<b>Unclarified points</b>	

<b>Use Case Name</b>	<b>Sell shares (Trade shares)</b>
<b>Person in charge</b>	
<b>Priority</b>	

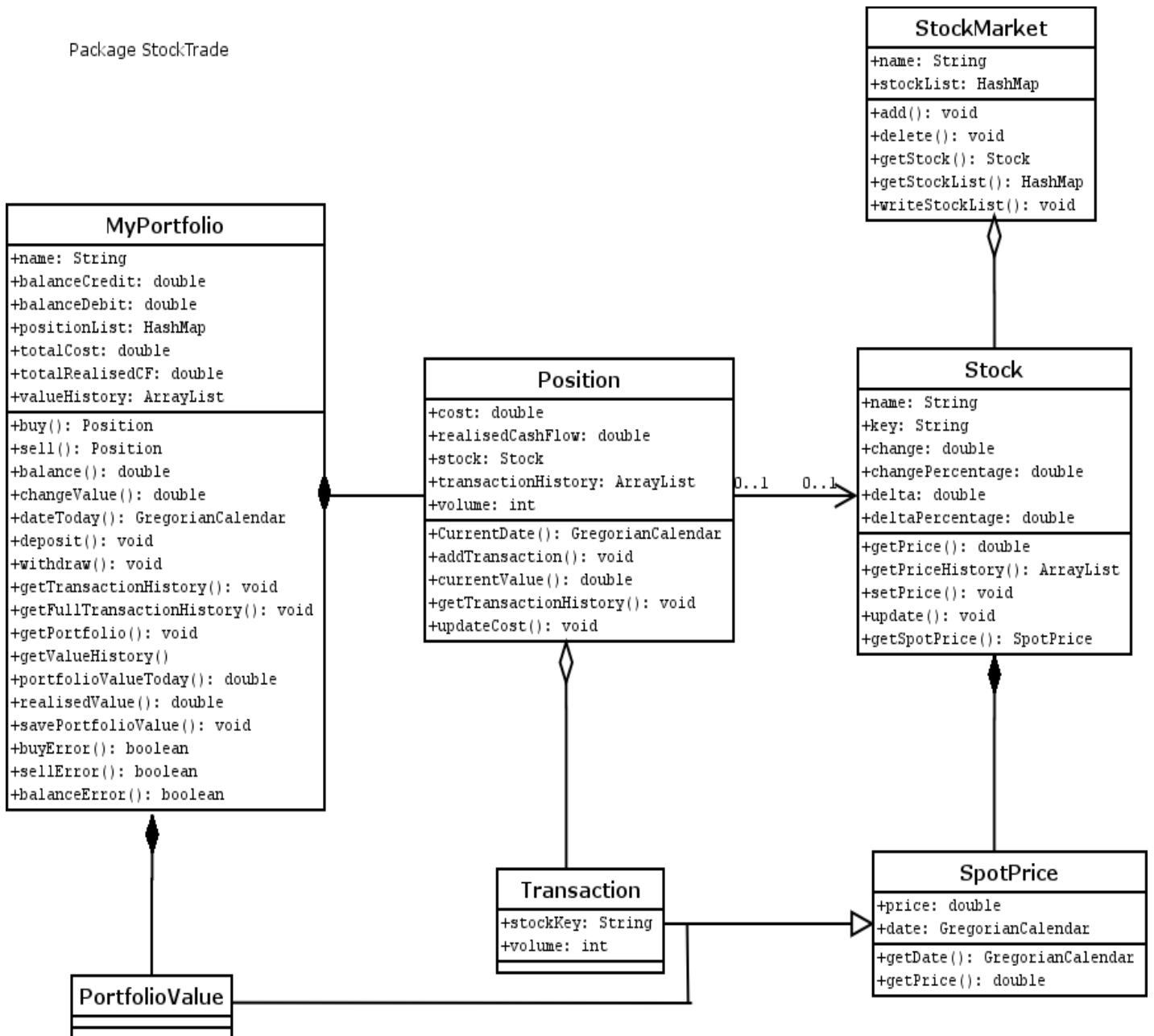
<b>Pre-Conditions</b>	<ul style="list-style-type: none"> <li>•JRE has been installed to run programme</li> <li>•Programme is activated</li> <li>•A new portfolio has been created</li> <li>•Portfolio contains one or more of the share to be sold</li> </ul>
<b>Post-Conditions</b>	<ul style="list-style-type: none"> <li>• <b>Application is open</b></li> <li>• <b>Trade does not reduce quantity of share held in portfolio to below zero</b></li> </ul>
<b>Trigger</b>	<ul style="list-style-type: none"> <li>•User clicks 'Sell' button on portfolio frame</li> </ul>
<b>Non-functional Boundary conditions</b>	
<b>Normal Flow</b>	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
<b>Error</b>	<ul style="list-style-type: none"> <li>• User does not feed in an integer variable as quantity</li> <li>• Quantity of share wished to be sold equals or is less than zero</li> <li>• Trade would reduce quantity of share held in portfolio to below zero</li> </ul>
<b>Result</b>	<ul style="list-style-type: none"> <li>• Portfolio table displays sale made</li> <li>• Displayed portfolio details updated to reflect sale</li> <li>• OR error message displayed</li> </ul>
<b>Notes</b>	NOTE: Short-selling not permitted
<b>Unclarified points</b>	

<b>Use Case Name</b>	<b>View portfolio position</b>
<b>Person in charge</b>	
<b>Priority</b>	

<b>Pre-Conditions</b>	<ul style="list-style-type: none"> <li>•JRE has been installed to run programme</li> <li>•Programme is activated</li> <li>•A new portfolio has been created</li> </ul>
<b>Post-Conditions</b>	<ul style="list-style-type: none"> <li>• <b>Application is open</b></li> </ul>
<b>Trigger</b>	<ul style="list-style-type: none"> <li>•User opens a new portfolio (Clicks 'Create Portfolio)</li> </ul>
<b>Non-functional Boundary conditions</b>	
<b>Normal Flow</b>	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
<b>Error</b>	<ul style="list-style-type: none"> <li>• Portfolio frame not opened according to instructions</li> </ul>
<b>Result</b>	<ul style="list-style-type: none"> <li>• Opened portfolio frame displays portfolio position details</li> </ul>
<b>Notes</b>	
<b>Unclarified points</b>	

### 2.3 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:

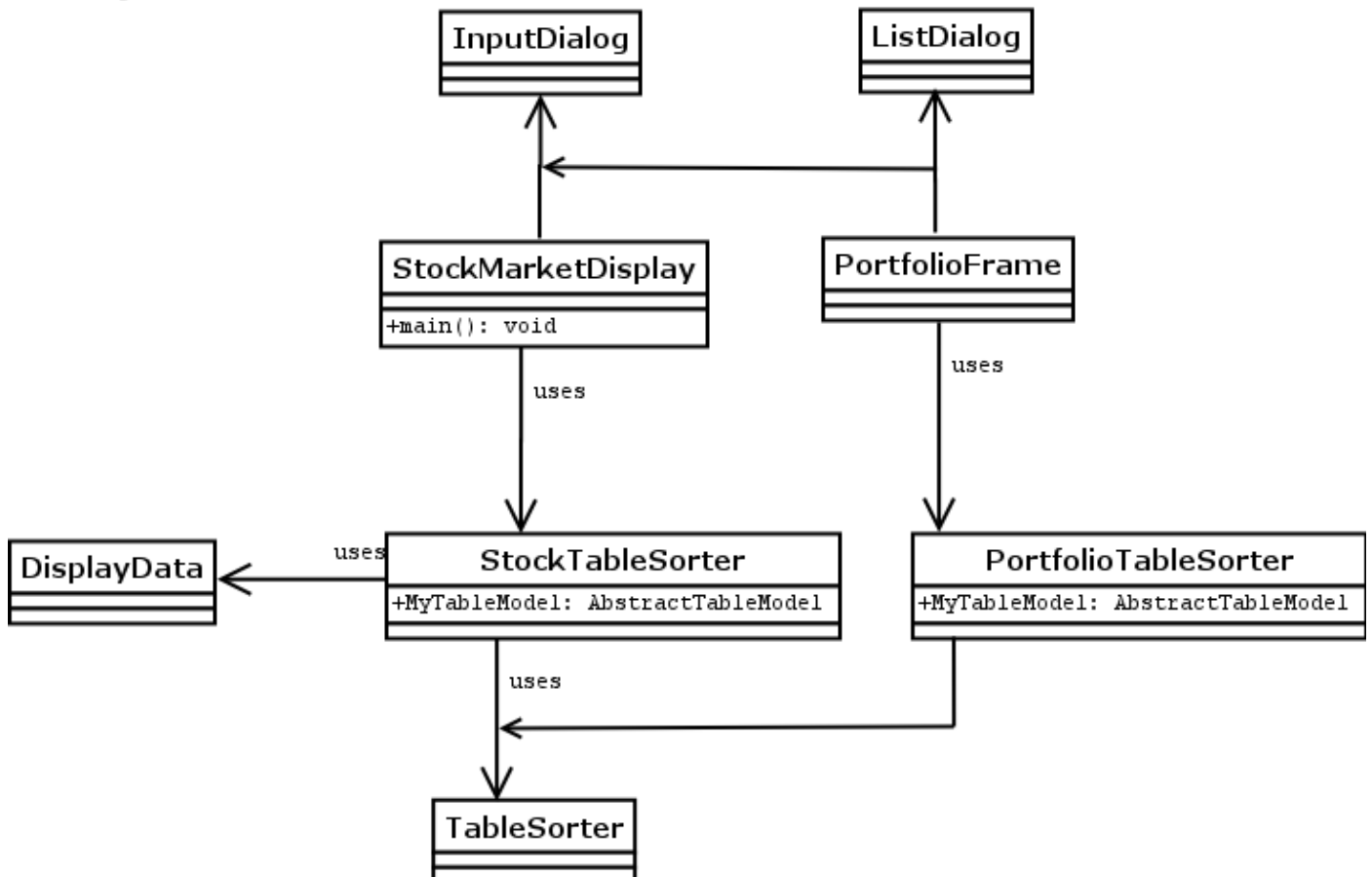
- StockMarket(String name): An instance of this class is a stock market where all the stocks are stored in a map. It has the following functions that are critical (used by GUI to display a table of all available stocks):

- 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 name, double balance) : 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
  - 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

Package GUI



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.



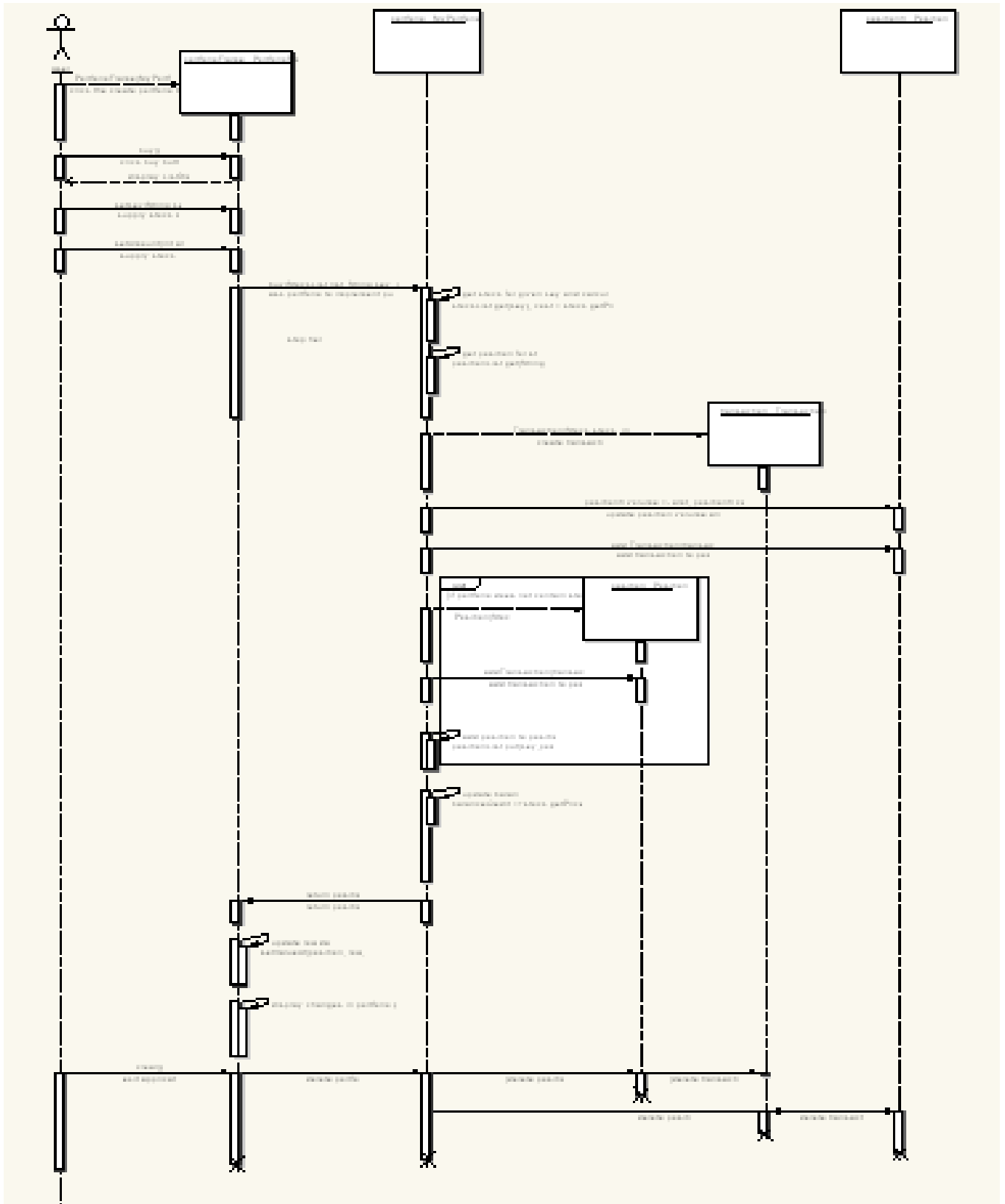
### Class Descriptions for Package GUIStockTrade:

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

- **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 first window to confront the user when the application is run and contains the main method for the application. 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.
- **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.

**2.4 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 buy(HashMap stockList, String stockKey, int qty)



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



The installation and use of this application on a computer requires the JRE (1.4 upwards) to be installed on this computer. The application is made available as a .jar file, which can be downloaded. In order to run the application, the user needs to open/run jar file. This can be done by :

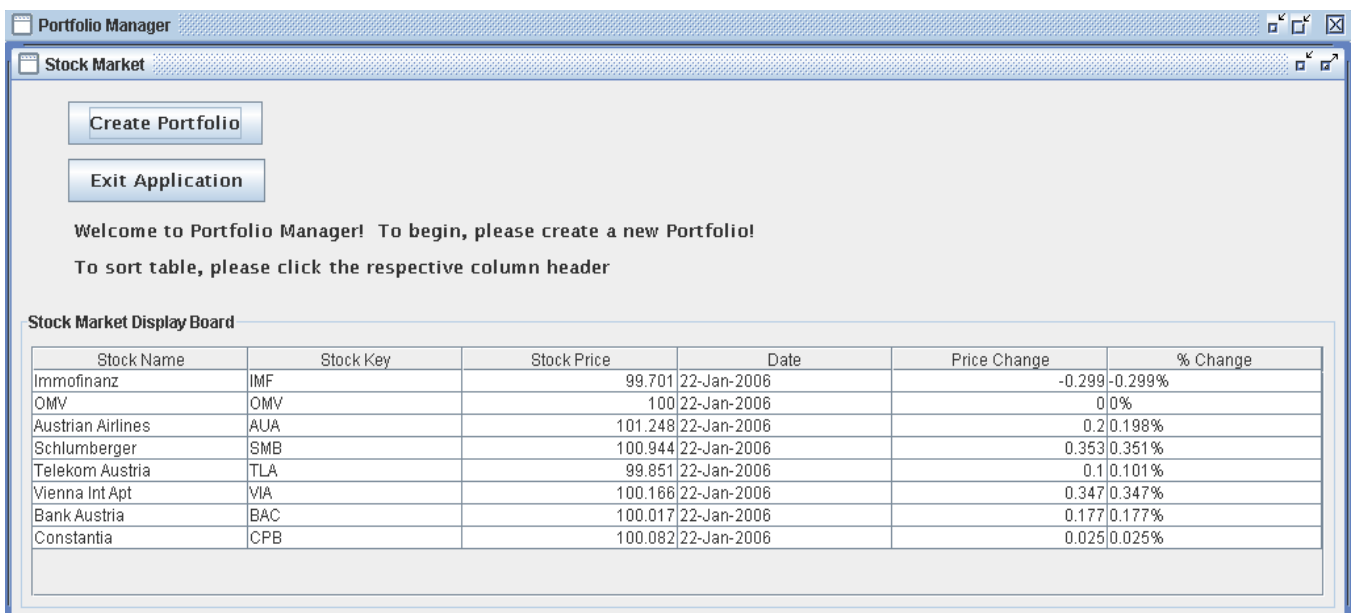


- double-clicking the file icon in a Windows operating system
- or by opening a command window and issuing the following command line:  
java -jar portfolioManager.jar

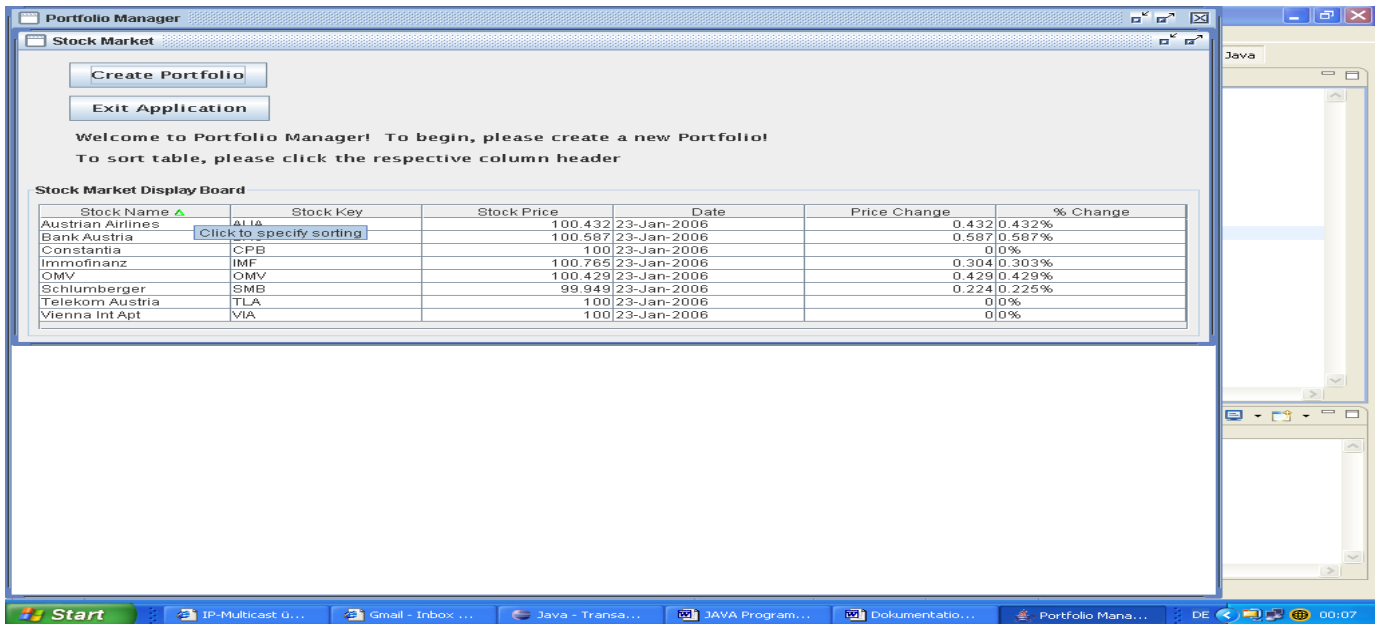
```
C:\Dokumente und Einstellungen\Shuba\Eigene Dateien>cd JAVA Praktikum
C:\Dokumente und Einstellungen\Shuba\Eigene Dateien\JAVA praktikum>java -jar portfolioManager.jar
Using font: Lucida Sans Unicode
C:\Dokumente und Einstellungen\Shuba\Eigene Dateien\JAVA praktikum>
```

It is important to access the file from the folder in which it is saved.

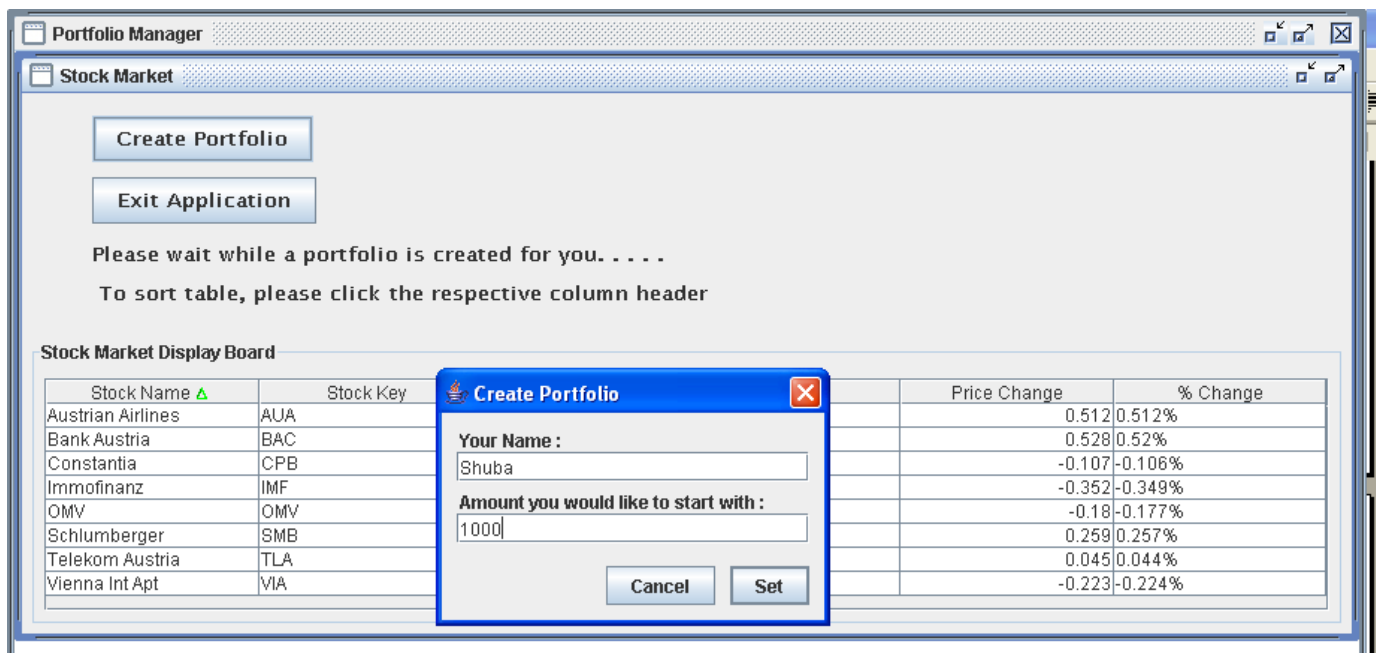
Upon opening, the following window is displayed. This is the stock market:



- To sort the stock market table, click any chosen column header



- To create a portfolio, click the 'Create Portfolio' button. Enter name and the balance to start with in the 'Create Portfolio' dialog box:



- A new portfolio window is created and the user can begin transacting in shares:

**Portfolio Manager**

**Stock Market**

Portfolio frame activated  
 To sort table, please click the respective column header

**Stock Market Display Board**

Stock Name ▲	Stock Key	Stock Price	Date	Price Change	% Change
Austrian Airlines	AUA	104.033	29-Jan-2006	-0.296	-0.284%
Bank Austria	BAC	109.177	29-Jan-2006	0.281	0.258%
Constantia	CPB	108.207	29-Jan-2006	-0.199	-0.184%
Immofinanz	IMF	105.47	29-Jan-2006	-0.084	-0.08%
OMV	OMV	107.312	29-Jan-2006	-0.398	-0.369%
Schlumberger	SMB	107.813	29-Jan-2006	-0.372	-0.344%
Telekom Austria	TLA	106.253	29-Jan-2006	0.367	0.347%
Vienna Int Apt	VIA	104.833	29-Jan-2006	-0.342	-0.325%

---

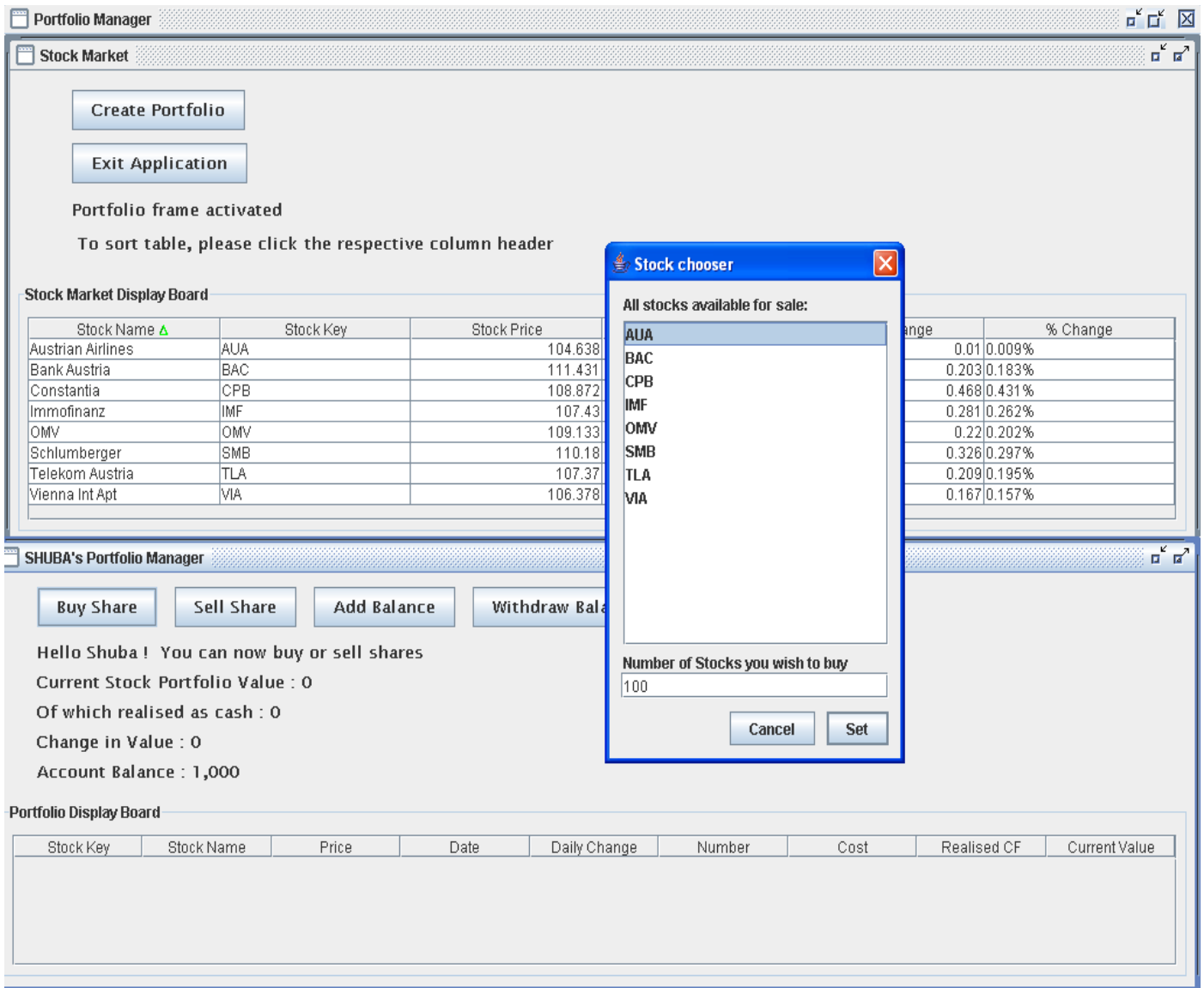
**SHUBA's Portfolio Manager**

Hello Shuba ! You can now buy or sell shares  
 Current Stock Portfolio Value : 0  
 Of which realised as cash : 0  
 Change in Value : 0  
 Account Balance : 1,000

**Portfolio Display Board**

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

- To buy shares, click the 'Buy Share' button. Choose a share and enter the number of shares you wish to buy:



- Upon purchase, the portfolio table as well as the portfolio position details are updated:

Portfolio Manager

Stock Market

Create Portfolio

Exit Application

Portfolio frame activated

To sort table, please click the respective column header

Stock Market Display Board

Stock Name ▲	Stock Key	Stock Price	Date	Price Change	% Change
Austrian Airlines	AUA	104.895	29-Jan-2006	-0.024	-0.023%
Bank Austria	BAC	113.127	29-Jan-2006	0.126	0.112%
Constantia	CPB	109.788	29-Jan-2006	0.553	0.506%
Immofinanz	IMF	111.115	29-Jan-2006	0.048	0.044%
OMV	OMV	108.912	29-Jan-2006	0.282	0.26%
Schlumberger	SMB	111.583	29-Jan-2006	0.28	0.252%
Telekom Austria	TLA	106.842	29-Jan-2006	-0.21	-0.196%
Vienna Int Apt	VIA	108.363	29-Jan-2006	0.204	0.189%

SHUBA's Portfolio Manager

Buy Share Sell Share Add Balance Withdraw Balance Market Tips

Last Purchase: 100 shares of Austrian Airlines! Please click the respective column header to sort table

Current Stock Portfolio Value : 104.895

Of which realised as cash : 0

Change in Value : 0.029

Account Balance : 895.134

Portfolio Display Board

Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value
AUA	Austrian Airlines	104.895	29-Jan-2006	0	100	10,486.62	0	10,489.513

- Similarly, to sell shares, click the 'Sell Share' button. Choose a share and enter the number of shares you wish to sell:

Portfolio Manager

Stock Market

Create Portfolio

Exit Application

Portfolio frame activated

To sort table, please click the respective column header

Stock Market Display Board

Stock Name ▲	Stock Key	Stock Price
Austrian Airlines	AUA	105.374
Bank Austria	BAC	114.658
Constantia	CPB	113.155
Immofinanz	IMF	111.433
OMV	OMV	109.889
Schlumberger	SMB	111.728
Telekom Austria	TLA	107.302
Vienna Int Apt	VIA	110.849

SHUBA's Portfolio Manager

Buy Share Sell Share Add Balance Withdraw Balance

Last Purchase: 100 shares of Austrian Airlines! Please click the res

Current Stock Portfolio Value : 105.374

Of which realised as cash : 0

Change in Value : 0.507

Account Balance : 895.134

Portfolio Display Board

Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value
AUA	Austrian Airlines	105.374	29-Jan-2006	0	100	10,486.62	0	10,537.362

Stock chooser

All stocks available for sale:

AUA

Number of Stocks you wish to sell

10

Cancel Set

- Upon sale, the portfolio table as well as the portfolio position details are updated:

Portfolio Manager

Stock Market

Create Portfolio

Exit Application

Portfolio frame activated

To sort table, please click the respective column header

Stock Market Display Board

Stock Name ▲	Stock Key	Stock Price	Date	Price Change	% Change
Austrian Airlines	AUA	106.003	29-Jan-2006	-0.091	-0.086%
Bank Austria	BAC	115.951	29-Jan-2006	0.381	0.33%
Constantia	CPB	113.516	29-Jan-2006	0.166	0.146%
Immofinanz	IMF	112.081	29-Jan-2006	0.009	0.008%
OMV	OMV	110.494	29-Jan-2006	-0.134	-0.121%
Schlumberger	SMB	113.246	29-Jan-2006	0.482	0.427%
Telekom Austria	TLA	108.841	29-Jan-2006	0.599	0.553%
Vienna Int Apt	VIA	111.428	29-Jan-2006	0.212	0.191%

SHUBA's Portfolio Manager

Buy Share   Sell Share   Add Balance   Withdraw Balance   Market Tips

Last Sale: 10 shares of Austrian Airlines! Please click the respective column header to sort table!

Current Stock Portfolio Value : 105.94

Of which realised as cash : 10.537

Change in Value : 1.074

Account Balance : 905.671

Portfolio Display Board

Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value
AUA	Austrian Airlines	106.003	29-Jan-2006	0	90	9,437.958	1,053.736	9,540.301

- To add to account balance, click the 'Add Balance' button. Enter amount to be added:

Portfolio Manager

Stock Market

Create Portfolio

Exit Application

Portfolio frame activated

To sort table, please click the respective column header

Stock Market Display Board

Stock Name ▲	Stock Key	Stock Price	Date	Price Change	% Change
Austrian Airlines	AUA	105.53	29-Jan-2006	-0.277	-0.261%
Bank Austria	BAC	116.083	29-Jan-2006	0.162	0.14%
Constantia	CPB	113.516	29-Jan-2006	0.166	0.146%
Immofinanz	IMF	112.346	29-Jan-2006	0.072	0.064%
OMV	OMV			0.283	0.255%
Schlumberger	SMB			-0.308	-0.27%
Telekom Austria	TLA			-0.221	-0.203%
Vienna Int Apt	VIA			0.182	0.164%

SHUBA's Portfolio Manager

Buy Share Sell Share Add Balance Withdraw Balance Market Tips

Last Sale: 10 shares of Austrian Airlines! Please click the respective column header to sort table!

Current Stock Portfolio Value : 105.438

Of which realised as cash : 10.537

Change in Value : 0.572

Account Balance : 905.671

Portfolio Display Board

Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value
AUA	Austrian Airlines	105.446	29-Jan-2006	0	90	9,437.958	1,053.736	9,490.104

- Upon deposit, the portfolio position details are updated:

Portfolio Manager

Stock Market

Create Portfolio

Exit Application

Portfolio frame activated

To sort table, please click the respective column header

Stock Market Display Board

Stock Name ▲	Stock Key	Stock Price	Date	Price Change	% Change
Austrian Airlines	AUA	107.914	29-Jan-2006	0.116	0.107%
Bank Austria	BAC	118.072	29-Jan-2006	0.187	0.159%
Constantia	CPB	116.223	29-Jan-2006	-0.232	-0.199%
Immofinanz	IMF	115.14	29-Jan-2006	0.225	0.196%
OMV	OMV	113.351	29-Jan-2006	0.382	0.338%
Schlumberger	SMB	116.734	29-Jan-2006	0.277	0.238%
Telekom Austria	TLA	110.18	29-Jan-2006	0.067	0.061%
Vienna Int Apt	VIA	110.758	29-Jan-2006	-0.206	-0.186%

SHUBA's Portfolio Manager

Buy Share Sell Share Add Balance Withdraw Balance Market Tips

Last Deposit: 100! Please click the respective column header to sort table!

Current Stock Portfolio Value : 107.66

Of which realised as cash : 10.537

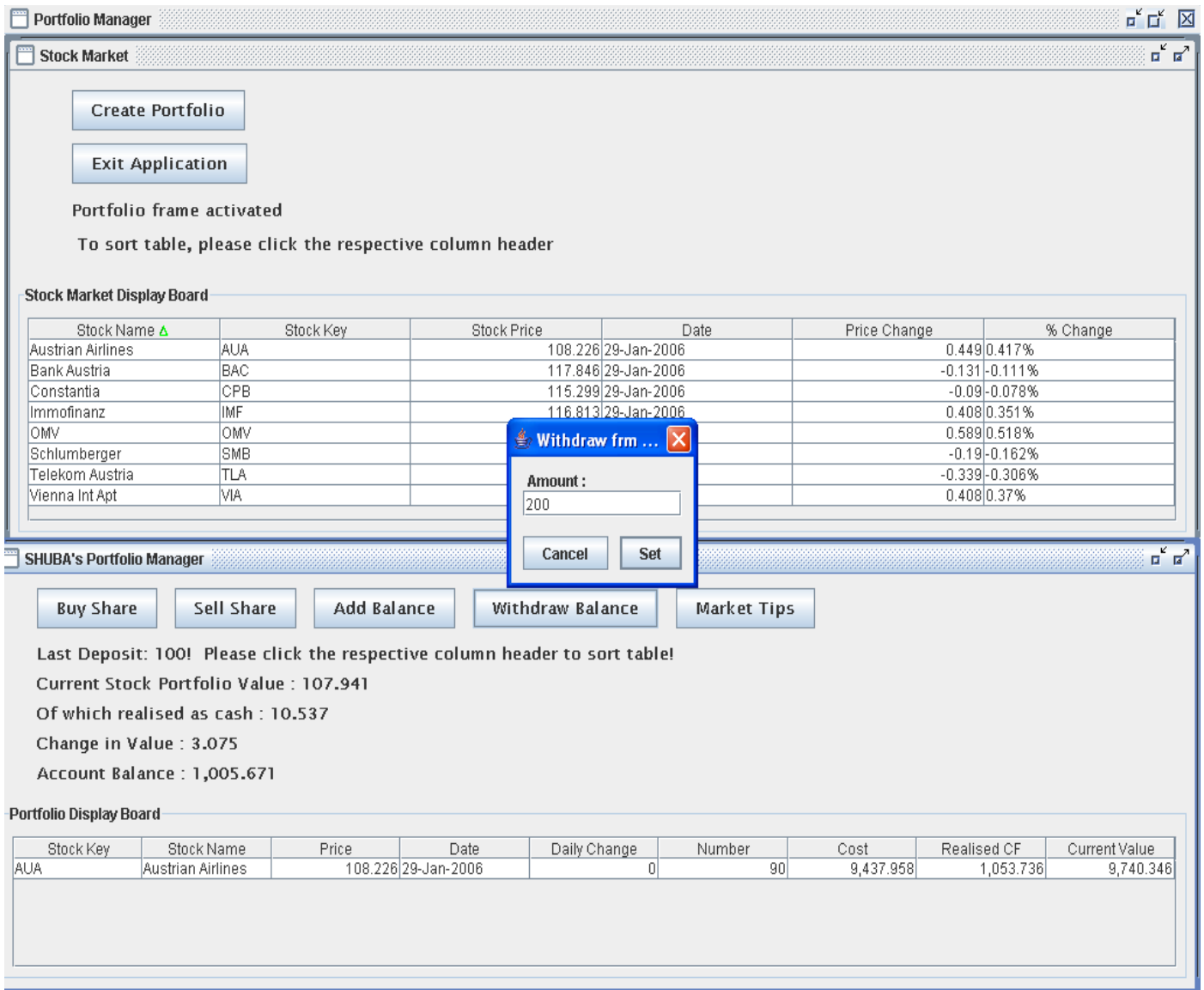
Change in Value : 2.794

Account Balance : 1,005.671

Portfolio Display Board

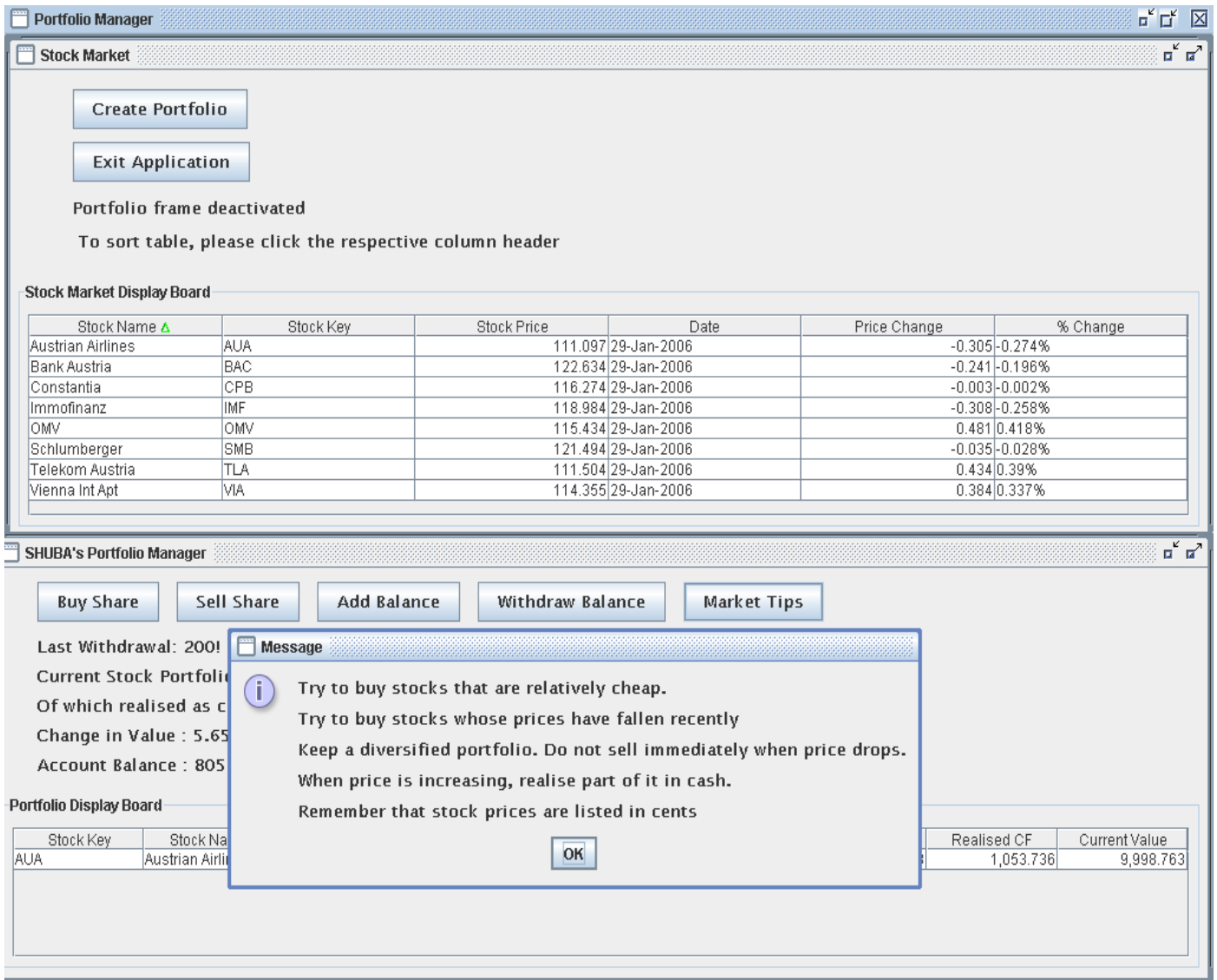
Stock Key	Stock Name	Price	Date	Daily Change	Number	Cost	Realised CF	Current Value
AUA	Austrian Airlines	107.914	29-Jan-2006	0	90	9,437.958	1,053.736	9,712.272

- To withdraw money, click the 'Withdraw Balance' button:



- Upon withdrawal, portfolio balance is updated in a similar fashion to the deposit procedure

- In order to obtain market tips, click the 'Market Tips' button:



- To exit application, click the 'Exit Application' button or close window.

#### 4 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.

## **5 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 140 hours, assuming 11 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

No formal model (such as COCOMO) was used to estimate the time required for project completion. Since no finished project could be used as an example, estimating lines of code (LOC) would have proven to be very erroneous (this is especially true of the GUI). Hence, it is uncertain whether such a model would have helped to finish the project on time.

The project itself was implemented as an iterative process and would most closely resemble the spiral model. This is mainly due to the learning curve involved.

Implementing the core model (analysis, design and implementation for StockTrade) required only 50 hours while the time expected for its implementation was 60 hours.

Implementing the GUI (analysis, design and implementation for GUI) required 80 hours (i.e., 30 hours more than expected). This was mainly due to the lack of experience with user interfaces and hence, steep learning curve involved (most of the code is self-written).

Integration and testing was fairly simple given that the project was implemented as an iterative process with integration and testing taking place after each step. Accordingly, the final integration and testing required only 15 hours (5 less than planned).

The documentation required a further 20 hours (expected: 10 hours). This was mainly due to administrative problems with different drawing tools and modelling packages.

The project can be summed up as follows:

Process	Expected Time in Hours	Incurred Time in Hours	Difference
Package StockTrade (Analysis, Design, Implementation)	60	50	-10
Package GUI (Analysis, Design, Implementation)	50	80	30
Integration and Testing	20	15	-5
Documentation	10	20	10
Total	140	165	25

Hence, the project overran by 25 hours; assuming 11 productive hours a week (12 weeks) and hence around 1.6 hours a day, the project required around 16 days more than scheduled. It should also be noted that in spite of the overrun, many of the functionalities conceived at the start were not implemented. This is mainly attributable to lack of time and the required expertise.

## 6 Bibliography

- Hahsler, M: Lecture slides for Introduction to Java Programming, <http://www.wu-wien.ac.at/~hahsler/JAVA/>
- Hahsler, M: Lecture slides for Java Programming Practical, [http://www.wu-wien.ac.at/~hahsler/JAVA\\_praktikum/](http://www.wu-wien.ac.at/~hahsler/JAVA_praktikum/)
- Sun GUI Tutorial: <http://java.sun.com/docs/books/tutorial/uiswing/>
- Stock market example: <http://www.java2s.com/ExampleCode/Swing-JFC/AnapplicationthatdisplaystockmarketdatainaJTable.htm>
- Violet, S and Walrath, K: 'Christmas Tree Applications: How to Create Frequently Updated JTables that Perform Well', <http://java.sun.com/products/jfc/tsc/articles/ChristmasTree/>
- Class TableSorter (package GUI) Source: <http://java.sun.com/docs/books/tutorial>
- Method getAFont() in DisplayData (Package GUI) Source: <http://java.sun.com/docs/books/tutorial>