

SYSTEMS PROGRAMMING AND SCRIPTING

FLORIAN BERGMANN

Assessment One: Stock Manager

CONTENTS

I DEVELOPMENT OF A STOCK MANAGER APPLICATION	1
1 INTRODUCTION	2
1.1 Document overview	2
1.2 Remit	2
2 REQUIREMENT'S CHECKLIST	4
3 DESIGN CONSIDERATIONS	5
3.1 Architectural overview	5
3.2 User Interface	6
3.3 Application Logic	7
4 USER GUIDE	8
4.1 Manage stock items and bank accounts	8
4.2 Placing an order	9
4.3 Importing & exporting data	10
4.3.1 File menu	10
4.3.2 Menu-bar icon	10
5 DEVELOPER GUIDE	12
5.1 UserInterface	12
5.2 ApplicationLogic	14
5.2.1 File handler	15
5.2.2 Error handling	15
6 TESTING	18
7 CONCLUSIONS	21
II APPENDIX	22
A APPENDIX: SOURCE CODE	23
A.1 View	23
A.2 Application Logic	36
A.2.1 Interfaces Package	36
A.2.2 Model Package	39
A.2.3 Presenter Package	60
A.3 Tests	67
BIBLIOGRAPHY	76

LIST OF FIGURES

Figure 1	Architecture overview with model-view-presenter (MVP) . . .	5
Figure 2	Abstract overview of project application logic	7
Figure 3	Main Window	8
Figure 4	Add a stock item.	9
Figure 5	Add a bank account.	9
Figure 6	Depositing and withdrawing money	9
Figure 7	Selection of items.	9
Figure 8	Placing an order without the needed funds	10
Figure 9	Saving via file menu	10
Figure 10	Settings window	11
Figure 11	Menu icon to save both lists.	11
Figure 12	Sequence diagram of input validation	17
Figure 13	NUnit test case run	20

LIST OF TABLES

Table 1	Performed tests.	19
---------	--------------------------	----

LISTINGS

Listing 1	Data Binding of view and model	12
Listing 2	Example interface IStockItemView	13
Listing 3	Interface ICongregateView	13
Listing 4	Interface ICSVSerializable	15
Listing 5	Validate method of StockItem	15

Listing 6	MainWindow.cs	23
Listing 7	Settings.cs	35
Listing 8	IBankAccountView.cs	36
Listing 9	ICongregateView.cs	37
Listing 10	ICSVSerializable.cs	37
Listing 11	IStockItemView.cs	38
Listing 12	IViewModel.cs	38
Listing 13	AppDataManager.cs	39
Listing 14	BankAccount.cs	46
Listing 15	ErrorMessage.cs	51
Listing 16	ErrorMessageCollection.cs	51
Listing 17	FileHandler.cs	52
Listing 18	NoFilePathSetException.cs	54
Listing 19	NotEnoughFundsException.cs	54
Listing 20	StockItem.cs	54
Listing 21	CongregatePresenter.cs	60
Listing 22	BankAccountTest.cs	67
Listing 23	FileHandlerTest.cs	70
Listing 24	StockItemTest.cs	70
Listing 25	AppDataManagerTest.cs	73

ACRONYMS

CSV	comma-separated values
GUI	graphical user interface
MBA	Management of Bank Accounts
MDA	Management of Data Access
MSI	Management of Stock Items
MVP	model-view-presenter
TDD	test-driven development

Part I

DEVELOPMENT OF A STOCK MANAGER APPLICATION

INTRODUCTION

In this chapter an overview over the document, as well as the specified requirements shall be given.

1.1 DOCUMENT OVERVIEW

This report fulfils in major parts the role of a requirements document. As such, it is intended for different audiences: [Chapter 2](#) provides an overview over the fulfilled requirements and thus should be of greatest interest for the managerial department, as well as the end users.

[Chapter 4](#) is a user guide that showcases the use of the program by showing how to accomplish certain tasks with the application. This part is essential for end users.

[Chapter 3](#) and [Chapter 5](#) are intended for engineers and software developers. They provide an overview over the application's high- and low-level design, highlighting certain important aspects that might need to be taken into account to allow further development to proceed at an efficient pace.

[Chapter 6](#) provides an overview over the testing that has happened during the development.

[Chapter 7](#) will wrap up the development of the application and provide an outlook at possible improvements that might be made.

1.2 REMIT

This section shall provide a short recap of the specified requirements. A list of fulfilled requirements will be provided in [Chapter 2](#).

The requirements, as understood by the contractor, are as follows ¹:

MSIO1: Allow the management of *stock items*. Management includes the following operations: *add*, *edit*, *delete*.

MSIO2: The operation *add* and *delete* should be possible without the use of an external storage.

¹ For further reference the requirements are prefixed with unique numbers: Management of Stock Items ([MSI](#)), Management of Bank Accounts ([MBA](#)), Management of Data Access ([MDA](#)), graphical user interface ([GUI](#))

MSIO3: Every stock item should consist of the following attributes: a *Stock Code*, an *item name*, a *supplier name*, a *unit's cost*, the *number required* and the *current stock*².

MSIO4: Allow the ordering of stock items via a money transfer.

MBA01: Allow the management of *bank accounts*: Management includes the following operations: *add*, *edit*, *delete*.

MBA02: The real transaction of money needs **not** to be implemented.

MBA03: An order should deduct the needed money from the bank account and change the *required* and *current* stock of an item accordingly.

MDA01: Allow the import and export of *stock items* from comma-separated values (CSV)-file.

MDA02: The location of the file may be chosen by the user.

MDA03: The ordering of the CSV-file may not be changed.

MDA04: The ordering of the file is as follows:

1	StockCode,Name,SupplierName,UnitCost,RequiredStock,CurrentStock
---	---

MDA04: The file should support blank fields by not entering data between two commas.

MDA05: The application should be able to handle *at least* 100 items.

GUI01: Interaction between user and program shall happen via a GUI.

GUI02: The GUI shall provide menus, buttons and icons for easier accessibility.

² For verification purposes it was assumed that the stock code is 4-digit number with trailing zeros allowed.

REQUIREMENT'S CHECKLIST

From the requirements stated in [Section 1.2](#), the following were fulfilled:

- MSIO1: Implemented in StockItem class with getters and setters.
- MSIO2: Implemented in a manager-class that handles adding and deleting in-memory.
- MSIO3: Implemented in StockItem class.
- MSIO4: Implemented in AppDataManager-class.
- MBAO1: Implemented in BankAccount class with getters and setters.
- MBAO2: Fake-method for ordering: adjusts account balance, without transfer money.
- MBAO3: Implemented in AppDataManager-class.
- MDAO1: Implemented in FileHandler-class and StockItem-class.
- MDAO2: Implemented in FileHandler-class.
- MDAO3: Implemented in StockItem-class.
- MDAO4: Implemented in StockItem-class.
- MDAO4: Implemented in StockItem-class.
- MDAO5: Verified via testing.
- GUIO1: Implemented via WinForms.
- GUIO2: Implemented via WinForms.

Apart from fulfilling these requirements the following features were implemented as well, to improve the user-experience of the program:

- ERROR NOTIFICATION: Upon entering invalid information the user will be informed about the mistakes by the [GUI](#).
- BANK ACCOUNT PERSISTENCE: It is possible to import and export bank accounts as well.
- ORDER QUANTITY: It is possible to order a certain quantity instead of always ordering the required number of items.

DESIGN CONSIDERATIONS

This chapter should provide a very general overview over the developed system, mainly describing the employed architecture.

3.1 ARCHITECTURAL OVERVIEW

The application was developed taking into account the principle of separating the program logic from interface design. To support this approach the model-view-presenter (MVP) design pattern was utilised.

In this pattern the presenter separates the GUI from the logical part of the application. The view communicates with the model only through the presenter. However, the model can notify the view directly of data-changes if an observer-pattern or data-binding is employed¹:

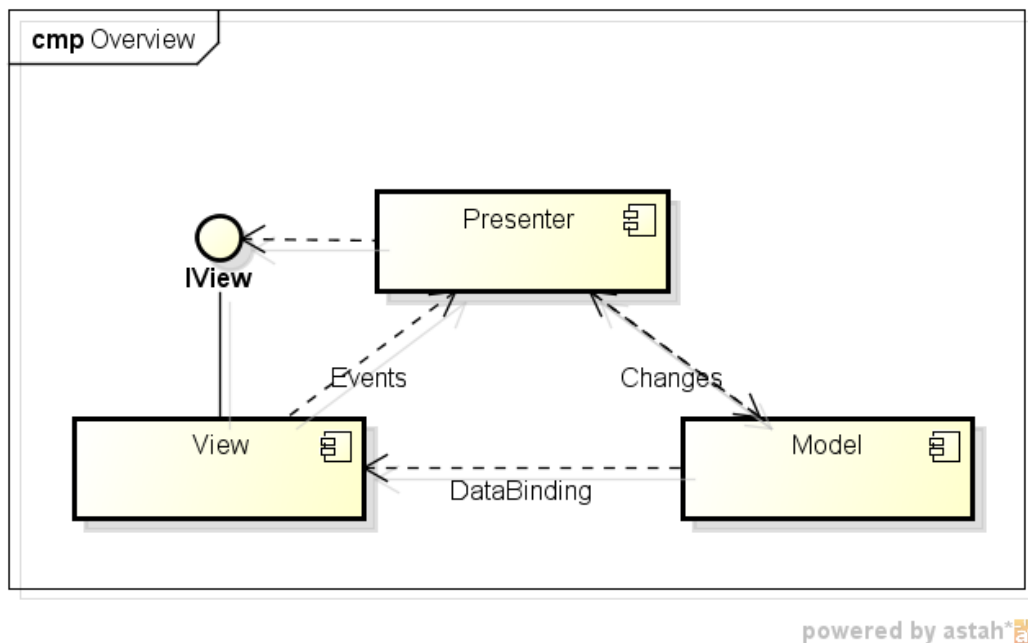


Figure 1: Architecture overview with MVP

¹ More information can be found at Boodhoo (2006).

Noteworthy is the `IView`-interface that allows the presenter to collect all needed data from the [GUI](#) without knowing what kind of [GUI](#) was used. This can help in reusing the presenter for multiple application-front-ends like WinForms or ASP.NET.

In the implementation part changes that occur in the model, will be forwarded to the view via the data-binding mechanisms provided by WinForms.

The implementation of this pattern splits the application into two projects:

USERINTERFACE: Hosts the graphical user interface and all code related to changing the appearance of the application.

APPLICATIONLOGIC: Hosts the presenter and the model component of the diagram.

Certain decisions made concerning these two packages will be described now, whereas greater detail will be put on implementation detail in [Chapter 5](#).

3.2 USER INTERFACE

The [GUI](#) was developed completely in WinForms utilizing only standard controls provided by the .NET framework.

To always display accurate data from the model, data-binding was used to connect the view to the model (further information about the concrete implementation can be found in [Section 5.1](#)).

The [GUI](#) project itself handles all changes to the [GUI](#)-elements (color changes, displaying new windows, etc.), whereas the collection of input-data from the controls is performed in the presenter via interfaces (more information in [Section 5.1](#)).

3.3 APPLICATION LOGIC

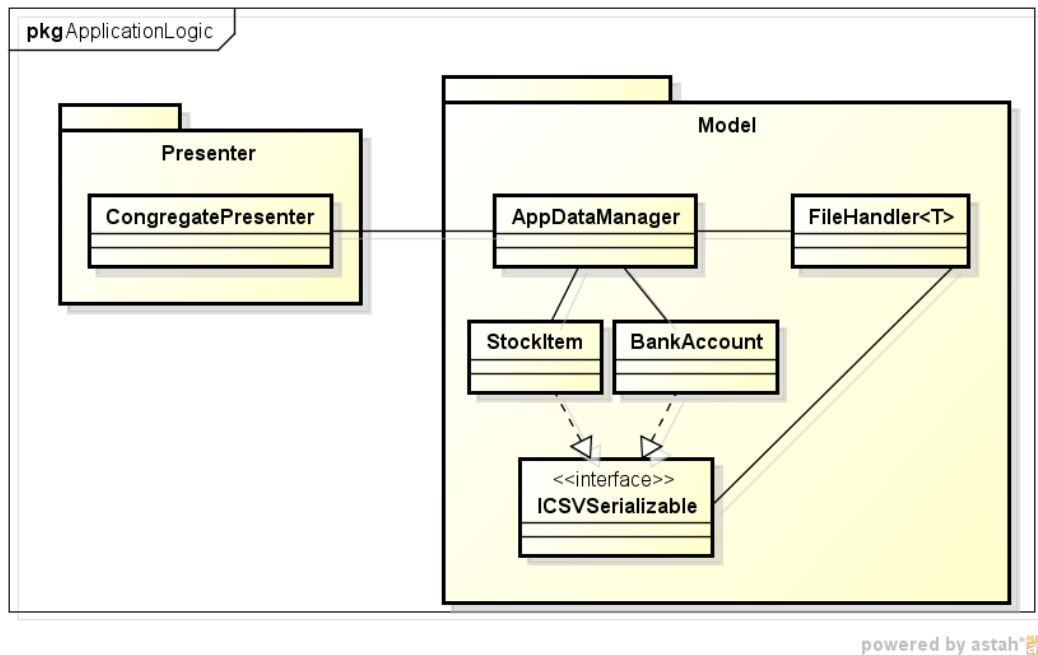


Figure 2: Abstract overview of project application logic

The application logic project was implemented in a very straight forward manner:

There is one class that handles all incoming requests - the *AppDataManager*. It coordinates statements as needed: e.g. check if enough money is present on bank account → place order → update stock item. Moreover the lists holding the stock items and bank accounts are managed by this class.

Naturally the classes for handling bank accounts and stock items are implemented in the application-logic-project as well. Moreover, a generic file handler (see [Section 5.2.1](#) for implementation details) and an error-handling-facility (see [Section 5.2.2](#)) were implemented.

Noteworthy is the fact that *BindingLists* were used in the *AppDataManager* to store the bank account and stock item lists. This was necessary to allow the data-binding with the view to work.

The *CongregatePresenter* seen in the picture is the connection point for the [GUI](#) part of the application and mostly forwards the requests to the *AppDataManager*.

USER GUIDE

In this chapter ways to achieve the most common use-cases of the program will be explained. These include:

1. Managing (adding, deleting, editing) a stock item or bank account.
2. Placing an order.
3. Importing and exporting data.

4.1 MANAGE STOCK ITEMS AND BANK ACCOUNTS

Upon starting the application the main window will be displayed. The main window hosts all necessary controls for the first two use cases.

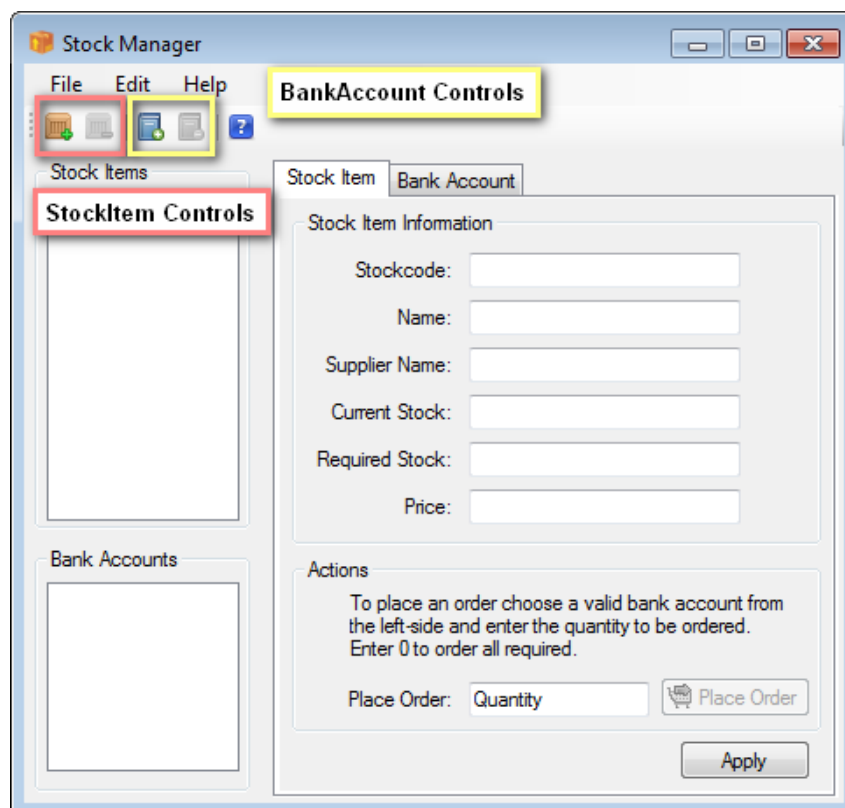


Figure 3: Main Window

To add a stock item or a bank account a click on the appropriate button is necessary:



Figure 4: Add a stock item.

By clicking the icon to add a new stock item to the application, an item will be inserted into the stock item list with dummy values.

By clicking the icon to add a new bank account to the application, a account will be inserted into the bank account list with dummy values.



Figure 5: Add a bank account.

After inserting a new stock item or bank account, the item can be chosen in the appropriate list (on the left-hand side of the application). By clicking an item, the appropriate panel will be show up, where the values can be edited.

Editing needs to be completed by clicking the *apply*-button. If any incorrect values were entered, the application will inform the user about the occurred mistakes.

To manage a bank account there are two more possible commands the user can issue: apart from changing the values, it is possible to deposit or withdraw money from the bank account. Therefore the user simply has to enter a number in the correct field and press the accompanying button.

The screenshot shows a panel titled 'Actions'. It contains two rows. The first row has the label 'Deposit:' followed by a text input field and a 'Deposit' button. The second row has the label 'Withdraw:' followed by a text input field and a 'Withdraw' button.

Figure 6: Depositing and withdrawing money

4.2 PLACING AN ORDER

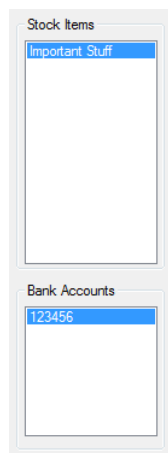


Figure 7: Selection of items.

To place an order the user has to select a bank account and a stock item from the lists (an item needs to be highlighted in both lists).

Then a value needs to be entered inside the *quantity*-box: either the amount of items to be ordered, or 0 (zero). By entering 0 the program will try to order the *required amount*.

If enough funds are available the order will be placed and the stock information will be updated. If not enough funds are available the application will output an error message.

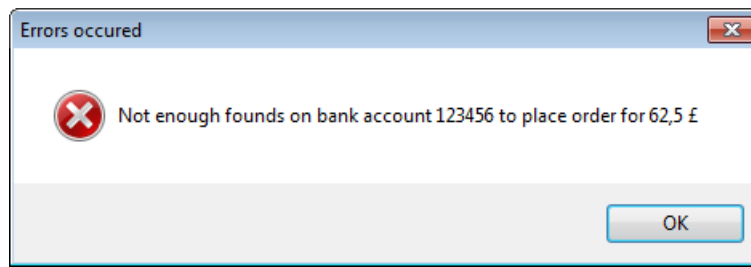


Figure 8: Placing an order without the needed funds

4.3 IMPORTING & EXPORTING DATA

After entering stock items and bank accounts it is possible to save them to a file and open them again for later use.

Therefore the user has to choose the appropriate option from the file menu or set standard-paths and click the menu-bar icon.

4.3.1 File menu

To save or load only one of the list the user selects File \Rightarrow Save (Open) \Rightarrow Save (Open) bank accounts / Save (Open) stock items (bank accounts).

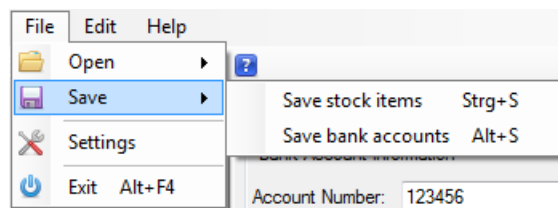


Figure 9: Saving via file menu

When loading a new set of items or accounts the currently stored objects will be discarded and overwritten by the contents of the file.

4.3.2 Menu-bar icon

To save via the menu icon it is necessary to first set default file paths for the files ¹. The paths can be set in the settings window found under File \Rightarrow Settings.

¹ As soon as these paths are set, the application will also attempt to load items and bank account on start-up.

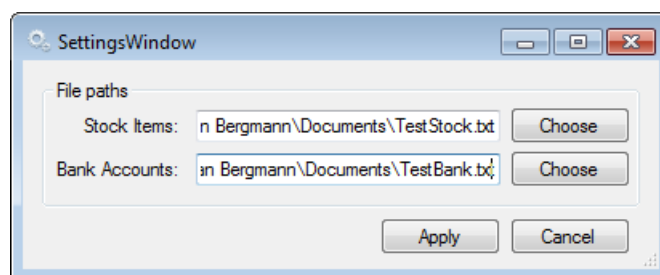


Figure 10: Settings window

After setting these paths both lists can be saved with a single click on the menu-bar icon (denoted by two disks):



Figure 11: Menu icon to save both lists.

To allow further development of the application, certain design decisions from [Chapter 3](#) will be described in greater depth:

5.1 USERINTERFACE

The user interface package holds the WinForms representation of a possible GUI¹.

The MainWindow holds a reference to the presenter and the model.

The presenter handles all events that need more logic than just changing the view's appearance².

The model-reference is used to set-up the data-binding in the application:

```

1 private void SetupDataBindings()
2     {
3         stockItemsListBox.DataSource = _Model.StockItems;
4         stockItemsListBox.DisplayMember = "Name";
5
6         /*
7          * The datasourceupdate mode is set to "Never".
8          * This leads to the ability to enforce the use of the presenter to update
9          * the values in the model.
10         * This way the validation errors can be handled by the presenter thus
11         * leading to better separation of concerns.
12         */
13         stockCodeTextBox.DataBindings.Add("Text", _Model.StockItems, "StockCode",
14             false, DataSourceUpdateMode.Never);
15         itemNameTextBox.DataBindings.Add("Text", _Model.StockItems, "Name", false,
16             DataSourceUpdateMode.Never);
17         supplierNameTextBox.DataBindings.Add("Text", _Model.StockItems, "
18             SupplierName", false, DataSourceUpdateMode.Never);
19         currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
20             ", false, DataSourceUpdateMode.Never);
21         reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
22             ", false, DataSourceUpdateMode.Never);
23         priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
24             , DataSourceUpdateMode.Never);
25
26         bankAccountsListBox.DataSource = _Model.BankAccounts;
27         bankAccountsListBox.DisplayMember = "AccountNumber";

```

¹ It is a *possible* GUI, as another one should - due to the decoupling of view and logic - be easily realisable by implementing the interfaces of the ApplicationLogic package.

² E.g. enabling/disabling buttons, changing the color of fields, showing a new window.


```

20
21     accountNumberTextBox.DataBindings.Add("Text", _Model.BankAccounts, "
        AccountNumber", false, DataSourceUpdateMode.Never);
22     nameTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Surname", false
        , DataSourceUpdateMode.Never);
23     balanceTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Balance",
        false, DataSourceUpdateMode.Never);
24 }

```

Listing 1: Data Binding of view and model

Noteworthy is the use of `DataSourceUpdateMode.Never`. This guarantees that changes from the GUI are not propagated to the model via data-binding, but that we can pass them through the presenter and keep the separation between view and model intact.

Another important architectural aspect of the view is the implementation of necessary interfaces for the presenter: instead of passing all attributes with a method call, the presenter will expect the view to implement a certain interface through which it can access the needed attributes:

```

1  i»using System;
2  namespace ApplicationLogic.Interfaces
3  {
4      /// <summary>
5      /// Utilized by the presenter to get the necessary values from a view.
6      /// </summary>
7      public interface IStockItemView
8      {
9          int CurrentStock { get; }
10         string ItemName { get; }
11         int RequiredStock { get; }
12         string StockCode { get; }
13         string SupplierName { get; }
14         double UnitCost { get; }
15     }
16 }

```

Listing 2: Example interface IStockItemView

The `MainWindow` implements three of these presenter-related interfaces: `IStockItemView`, `IBankAccountView`, `ICongregateView`. The first two guarantee the presenter that it can access all attributes needed to update an item or bank account. The later view provides the following methods and properties:

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using ApplicationLogic.Model;

```

```

6
7 namespace ApplicationLogic.Interfaces
8 {
9     /// <summary>
10    /// Utilized by the presenter to get the necessary values from a view.
11    /// </summary>
12    public interface ICongregateView
13    {
14        StockItem StockItem { get; }
15
16        BankAccount BankAccount { get; }
17
18        int Quantity { get; }
19
20        double Deposit { get; }
21
22        double Withdraw { get; }
23
24        bool ConfirmDelete();
25
26        bool ConfirmClose();
27
28        void DisplayValidationErrors(ErrorMessageCollection errorCollection);
29    }
30 }

```

Listing 3: Interface ICongregateView

It allows to delete items and bank accounts, provides the necessary application logic to order items, deposit and withdraw money and exposes a method to display possible validation errors.

Should new views be added an interface should be provided that guarantees the separation between view and presenter. This might allow the reuse of the presenter across multiple GUIs.

A problem arising from the .NET architecture is that only the GUI project provides a settings file. This leads to the fact that the WinForms-GUI has to handle the loading and saving of user-preferences (the file paths to the bank accounts and stock items files).

5.2 APPLICATIONLOGIC

The application logic project hosts the presenter as well as the models.

As has been pointed out in [Chapter 3](#), the AppDataManager-class works as a *facade* for the rest of the model.

Noteworthy implementations in this package are the file handler and the realisation of error handling.

5.2.1 File handler

The FileHandler-class utilizes the concept of generics: this way it is possible to reuse this class for multiple classes that need to be persisted.

To allow the serialization and de-serialization-logic to be separated from the file-handling logic, the FileHandler-class requires all classes that need to be persisted to implement the ICSVSerializable-interface:

```

1  >using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Interfaces
7  {
8      /// <summary>
9      /// Used to ensure all objects will be able to persisted via FileHandler class.
10     /// </summary>
11     /// <typeparam name="T"></typeparam>
12     public interface ICSVSerializable<T>
13     {
14         String CsvRepresentation();
15
16         T ParseFromString(String stringRepresentation);
17     }
18 }
```

Listing 4: Interface ICSVSerializable

Due to this fact - both - the StockItem and the BankAccount-class implement this interface.

5.2.2 Error handling

Error handling is achieved by the separate classes ErrorMessageCollection and ErrorMessage.

On validation, error-messages will be added to an error-message-collection that can be accessed from the presenter to force the view to display the occurred errors.

As an example the code of the stock item's Validate() method, as well as a sequence-diagram of the calling sequence is shown:

```

1  public static bool Validate(String stockCode, String name, String supplierName, double
    unitCost, int required, int currentStock)
2      {
3          if (String.IsNullOrEmpty(stockCode) || !IsValidStockCode(stockCode))
4          {
5              ErrorMessageCollection.Add(new ErrorMessage("Need a stockcode that adheres to
                the stockcode format: 4 numbers."));

```

```
6      }
7      if (String.IsNullOrEmpty(name))
8      {
9          ErrorMessage.Add(new ErrorMessage("Need an item name. "));
10     }
11     if (String.IsNullOrEmpty("supplierName"))
12     {
13         ErrorMessage.Add(new ErrorMessage("Need a supplier name. "));
14     }
15     if (unitCost < 0.0)
16     {
17         ErrorMessage.Add(new ErrorMessage("Unit costs must be greater or
18             equal 0. "));
19     }
20     if (required < 0)
21     {
22         ErrorMessage.Add(new ErrorMessage("Required must be greater or equal
23             0. "));
24     }
25     if (currentStock < 0)
26     {
27         ErrorMessage.Add(new ErrorMessage("Current must be greater or equal
28             0. "));
29     }
30     return ErrorMessage.Count == 0;
31 }
```

Listing 5: Validate method of StockItem

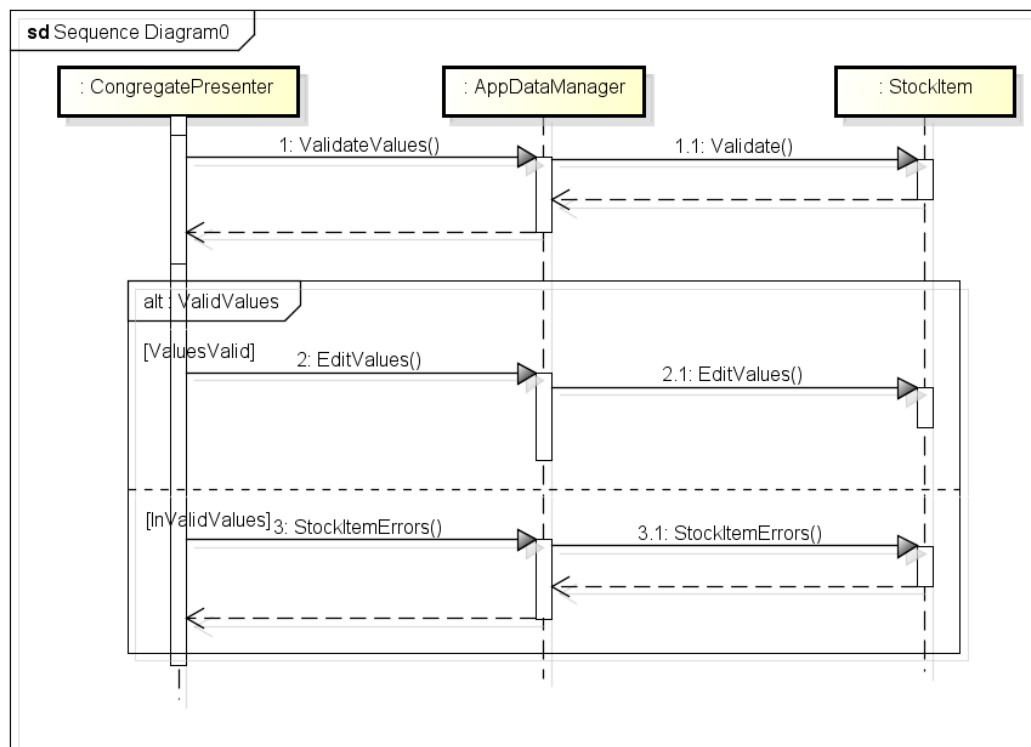


Figure 12: Sequence diagram of input validation

TESTING

Testing was performed in two stages:

1. Unit-Testing
2. System-Integration-Testing

The part of sub-system-integration-testing as lined out in Sommerville (2006, p. 520) was skipped, as there was no significant number of sub-modules that needed testing. Most of the integration consisted of passing on parameters through the three layers of the application (view, presenter, model).

These tests were incorporated into the main system-integration-testing.

As the model was the part that hosted most of the application logic, test-cases were created for its four main-classes:

1. StockItem
2. BankAccount
3. FileHandler
4. AppDataManager

The concrete error-cases that were tested are:

Under Test	Input	Output
Stock Item	Invalid StockCode	ArgumentException
Stock Item	Invalid Current Stock	ArgumentException
Stock Item	Invalid Required Stock	ArgumentException
Stock Item	Invalid Parsing	FormatException
Stock Item	Invalid Cost	ArgumentException
Bank Account	Invalid Balance	ArgumentException
Bank Account	Invalid Withdraw (Value too small)	ArgumentException
Bank Account	Invalid Withdraw (Value too high)	NotEnoughFundsException

Bank Account	Invalid Deposit (Value too small)	ArgumentException
Bank Account	Invalid Transfer (Value too small)	ArgumentException
Bank Account	Invalid Transfer (Value too high)	NotEnoughFundsException
Bank Account	Too little funds for transfer	NotEnoughFundsException
Bank Account	Invalid Parsing	FormatException
AppDataManager	Remove invalid item	ArgumentException
AppDataManager	Perform Order	Correct sequence of statements
AppDataManager	Order with too little funds	NotEnoughFundsException
File Handler	Save without file path	NoFilePathSetException
File Handler	Load without file path	NoFilePathSetException

Table 1: Performed tests.

Moreover test cases were written to ensure correct behaviour for correct input. A complete listing of all test cases can be found under [Section A.3](#)¹.

Upon delivery a comprehensive suite of passing test cases is provided:

¹ Running the test cases will require the [NUnit](http://www.nunit.org/) (<http://www.nunit.org/>)- and [Moq](http://code.google.com/p/moq/) (<http://code.google.com/p/moq/>)-libraries.

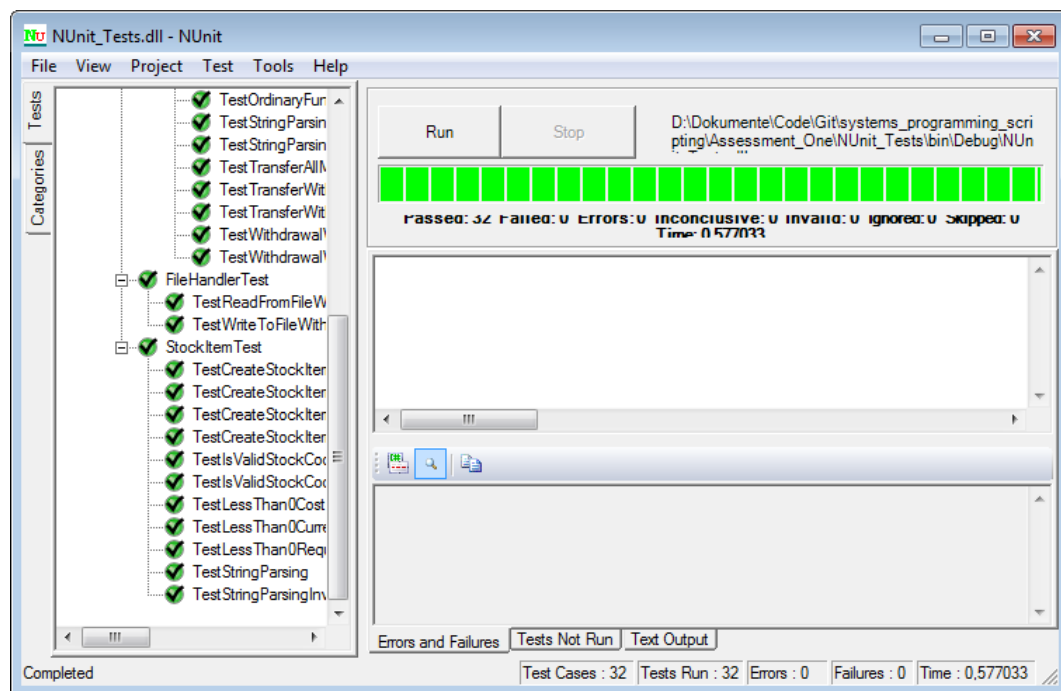


Figure 13: NUnit test case run

CONCLUSIONS

After developing the application and looking at the resulting design, there are a few things that might have been accomplished differently and produced a better and cleaner design.

The data-binding part - due to the removal of the back-write mechanism (so the view does not automatically update the model) was mostly unnecessary and restricted the choices of collections in the model-part to `BindingLists`, whereas the model should actually be independent of the view.

Another problem arising from the `BindingLists` is that they can not be returned as read-only and this way can provide write access to the model. However, the lists were still implemented as public properties to allow unit-testing.

Another fact that became clear while testing was that the auto-creation of stock items and bank accounts in the model was unfortunate to test the correct behaviour of adding and deleting items. One could not easily add a specific item and delete the same one, but let the model create one, acquire a reference to it, delete it and test if the list was changed accordingly. A thorough test-driven development ([TDD](#))-approach might have circumvented these design-shortcomings.

However, even with these shortcomings the application should fulfil the requirements. Moreover, the implementation of the generic file handler should be reusable to persist all kinds of objects - even in other projects.

And with the provided test-cases many cases should be covered that should allow a thorough refactoring of the application if the need arises.

Part II

APPENDIX

APPENDIX: SOURCE CODE

A.1 VIEW

```
1  i»using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Configuration;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Windows.Forms;
10 using ApplicationLogic.Interfaces;
11 using ApplicationLogic.Presenter;
12 using ApplicationLogic.Model;
13 using System.Collections.Specialized;
14 using Assessment_One.Properties;
15
16 namespace Assessment_One
17 {
18     public partial class MainWindow : Form, ICongregateView, IStockItemView,
        IBankAccountView
19     {
20         private CongregatePresenter _Presenter;
21         private IViewModel _Model;
22         private List<Control> backgroundColorChanged;
23
24         private const int STOCKITEMTAB = 0;
25         private const int BANKACCOUNTTAB = 1;
26
27         /// <summary>
28         /// Initializes a new instance of the <see cref="MainWindow"/> class.
29         /// </summary>
30         public MainWindow()
31         {
32             InitializeComponent();
33             this._Model = new AppDataManager();
34             _Presenter = new CongregatePresenter(this, this, this, this._Model);
35             this.backgroundColorChanged = new List<Control>();
36             LoadFilePathSettings();
37             SetUpDataBindings();
38         }
39
40     }
```

```

41
42 #region EventHandler
43
44 private void bankAccountsListBox_SelectedIndexChanged(object sender, EventArgs
45     e)
46 {
47     if (this.bankAccountsListBox.SelectedItem != null)
48     {
49         this.tabControl1.SelectTab(BANKACCOUNTTAB);
50         this.SwitchBankAccountControls(true);
51     }
52     else
53     {
54         this.SwitchBankAccountControls(false);
55     }
56 }
57
58 private void stockItemsListBox_SelectedValueChanged(object sender, EventArgs e
59     )
60 {
61     if (this.stockItemsListBox.SelectedItem != null)
62     {
63         this.tabControl1.SelectTab(STOCKITEMTAB);
64         this.SwitchStockItemControls(true);
65     }
66     else
67     {
68         this.SwitchStockItemControls(false);
69     }
70 }
71
72 private void addStockItemToolStripMenuItem_Click(object sender, EventArgs e)
73 {
74     this._Presenter.CreateNewStockItem();
75 }
76
77 private void deleteStockItemToolStripMenuItem_Click(object sender, EventArgs e
78     )
79 {
80     _Presenter.DeleteStockItem();
81     this.quantityTextBox.Text = "";
82 }
83
84 private void addBankAccountToolStripMenuItem_Click(object sender, EventArgs e)
85 {
86     this._Presenter.CreateNewBankAccount();
87 }
88
89 private void deleteBankAccountToolStripMenuItem_Click(object sender, EventArgs
90     e)
91 {

```

```

88         this._Presenter.DeleteBankAccount();
89         this.depositQuantityTextBox.Text = "";
90         this.withdrawQuantityTextBox.Text = "";
91     }
92
93     private void openStockItemToolStripMenuItem_Click(object sender, EventArgs e)
94     {
95
96         if (this.openFileDialog.ShowDialog() == DialogResult.OK)
97         {
98             OpenFileDialog file = this.openFileDialog;
99             this._Presenter.LoadStockItemsFromFile(file.FileName);
100         }
101     }
102
103     private void openBankAccountsToolStripMenuItem_Click(object sender, EventArgs
104         e)
105     {
106         if (this.openFileDialog.ShowDialog() == DialogResult.OK)
107         {
108             OpenFileDialog file = this.openFileDialog;
109             this._Presenter.LoadBankAccountsFromFile(file.FileName);
110         }
111     }
112
113     private void saveStockItemsToolStripMenuItem_Click(object sender, EventArgs e)
114     {
115         if (this.saveFileDialog.ShowDialog() == DialogResult.OK)
116         {
117             SaveFileDialog file = this.saveFileDialog;
118             this._Presenter.SaveStockItemsToFile(file.FileName);
119         }
120     }
121
122     private void saveBankAccountsToolStripMenuItem_Click(object sender, EventArgs
123         e)
124     {
125         if (this.saveFileDialog.ShowDialog() == DialogResult.OK)
126         {
127             SaveFileDialog file = this.saveFileDialog;
128             this._Presenter.SaveBankAccountsToFile(file.FileName);
129         }
130     }
131
132     private void applyButton_Click(object sender, EventArgs e)
133     {
134         this.ResetColoring();
135         this._Presenter.EditStockItem();
136     }
137
138     private void ResetColoring()

```

```

137     {
138         foreach (Control control in this.backgroundColorChanged)
139         {
140             control.BackColor = Color.White;
141         }
142     }
143
144     private void applyBankAccountButton_Click(object sender, EventArgs e)
145     {
146         this.ResetColoring();
147         this._Presenter.EditBankAccount();
148     }
149
150     void PlaceOrderButton_Click(object sender, EventArgs e)
151     {
152         this._Presenter.OrderItem();
153         this.quantityTextBox.Text = "";
154     }
155
156     private void depositButton_Click(object sender, EventArgs e)
157     {
158         this._Presenter.Deposit();
159         this.depositQuantityTextBox.Text = "";
160     }
161
162     private void withdrawButton_Click(object sender, EventArgs e)
163     {
164         this._Presenter.Withdraw();
165         this.withdrawQuantityTextBox.Text = "";
166     }
167
168     private void exitToolStripMenuItem_Click(object sender, EventArgs e)
169     {
170         this._Presenter.CloseApplication();
171     }
172
173     private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
174     {
175         SettingsWindow sw = new SettingsWindow();
176         sw.ShowDialog();
177         this.LoadFilePathSettings();
178     }
179
180     private void saveStripButton_Click(object sender, EventArgs e)
181     {
182         this._Presenter.SaveBankAccountsToFile();
183         this._Presenter.SaveStockItemsToFile();
184     }
185
186     private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
187     {

```

```

188         if (tabControl1.SelectedIndex == BANKACCOUNTTAB)
189         {
190             if (bankAccountsListBox.SelectedItem != null)
191             {
192                 this.SwitchBankAccountControls(true);
193             }
194             else
195             {
196                 this.SwitchBankAccountControls(false);
197             }
198         }
199         else
200         {
201             if (stockItemsListBox.SelectedItem != null)
202             {
203                 this.SwitchStockItemControls(true);
204             }
205             else
206             {
207                 this.SwitchStockItemControls(false);
208             }
209         }
210     }
211
212     private void quantityTextBox_TextChanged(object sender, EventArgs e)
213     {
214         CheckQuantity();
215     }
216
217     #endregion
218
219     #region IStockItemView
220
221     /// <summary>
222     /// Gets the current stock.
223     /// </summary>
224     /// <value>The current stock.</value>
225     public int CurrentStock
226     {
227         get
228         {
229             int currentStock;
230             try
231             {
232                 currentStock = int.Parse(currStockTextBox.Text);
233                 return currentStock;
234             }
235             catch (FormatException e)
236             {
237                 this.DisplayError(currStockTextBox);
238                 throw;

```

```

239         }
240     }
241 }
242
243 /// <summary>
244 /// Gets the required stock.
245 /// </summary>
246 /// <value>The required stock.</value>
247 public int RequiredStock
248 {
249     get
250     {
251         int requiredStock;
252         try
253         {
254             requiredStock = int.Parse(reqStockTextBox.Text);
255             return requiredStock;
256         }
257         catch (FormatException e)
258         {
259             this.DisplayError(reqStockTextBox);
260             throw;
261         }
262     }
263 }
264
265 /// <summary>
266 /// Gets the stock code.
267 /// </summary>
268 /// <value>The stock code.</value>
269 public string StockCode
270 {
271     get { return stockCodeTextBox.Text; }
272 }
273
274 /// <summary>
275 /// Gets the name of the supplier.
276 /// </summary>
277 /// <value>The name of the supplier.</value>
278 public string SupplierName
279 {
280     get { return supplierNameTextBox.Text; }
281 }
282
283 /// <summary>
284 /// Gets the unit cost.
285 /// </summary>
286 /// <value>The unit cost.</value>
287 public double UnitCost
288 {
289     get

```



```

290         {
291             double unitCost;
292             try
293             {
294                 unitCost = double.Parse(priceTextBox.Text);
295                 return unitCost;
296             }
297             catch (FormatException e)
298             {
299                 this.DisplayError(priceTextBox);
300                 throw;
301             }
302         }
303     }
304
305     /// <summary>
306     /// Gets the name of the item.
307     /// </summary>
308     /// <value>The name of the item.</value>
309     public string ItemName
310     {
311         get { return itemNameTextBox.Text; }
312     }
313
314     #endregion
315
316     #region IBankAccountView
317
318     /// <summary>
319     /// Gets the account number.
320     /// </summary>
321     /// <value>The account number.</value>
322     public int AccountNumber
323     {
324         get
325         {
326             int accountNumber;
327             try
328             {
329                 accountNumber = int.Parse(accountNumberTextBox.Text);
330                 return accountNumber;
331             }
332             catch (FormatException e)
333             {
334                 this.DisplayError(accountNumberTextBox);
335                 throw;
336             }
337         }
338     }
339
340     /// <summary>

```

```

341     /// Gets the surname.
342     /// </summary>
343     /// <value>The surname.</value>
344     public string Surname
345     {
346         get { return nameTextBox.Text; }
347     }
348
349     /// <summary>
350     /// Gets the balance.
351     /// </summary>
352     /// <value>The balance.</value>
353     public double Balance
354     {
355         get
356         {
357             double balance;
358             try
359             {
360                 balance = double.Parse(balanceTextBox.Text);
361                 return balance;
362             }
363             catch (FormatException e)
364             {
365                 this.DisplayError(balanceTextBox);
366                 throw;
367             }
368         }
369     }
370
371     #endregion
372
373     #region ICongregateView
374
375     /// <summary>
376     /// Gets the stock item.
377     /// </summary>
378     /// <value>The stock item.</value>
379     public StockItem StockItem
380     {
381         get { return (StockItem)this.stockItemsListBox.SelectedItem; }
382     }
383
384     /// <summary>
385     /// Gets the bank account.
386     /// </summary>
387     /// <value>The bank account.</value>
388     public BankAccount BankAccount
389     {
390         get { return (BankAccount)this.bankAccountsListBox.SelectedItem; }
391     }

```

```

392
393     /// <summary>
394     /// Gets the quantity value.
395     /// </summary>
396     /// <value>The quantity.</value>
397     public int Quantity
398     {
399         get
400         {
401             int quan = 0;
402             try
403             {
404                 quan = int.Parse(quantityTextBox.Text);
405                 return quan;
406             }
407             catch (FormatException e)
408             {
409                 this.DisplayError(quantityTextBox);
410                 throw;
411             }
412         }
413     }
414
415     /// <summary>
416     /// Gets the deposit value.
417     /// </summary>
418     /// <value>The deposit.</value>
419     public double Deposit
420     {
421         get
422         {
423             double deposit;
424             try
425             {
426                 deposit = double.Parse(depositQuantityTextBox.Text);
427                 return deposit;
428             }
429             catch (FormatException e)
430             {
431                 this.DisplayError(depositQuantityTextBox);
432                 throw;
433             }
434         }
435     }
436
437     /// <summary>
438     /// Gets the withdraw value.
439     /// </summary>
440     /// <value>The withdraw.</value>
441     public double Withdraw
442     {

```

```

443         get
444         {
445             double withdraw;
446             try
447             {
448                 withdraw = double.Parse(withdrawQuantityTextBox.Text);
449                 return withdraw;
450             }
451             catch (FormatException e)
452             {
453                 this.DisplayError(withdrawQuantityTextBox);
454                 throw;
455             }
456         }
457     }
458
459     /// <summary>
460     /// Displays the validation errors.
461     /// </summary>
462     /// <param name="errorCollection">The error collection.</param>
463     public void DisplayValidationErrors(ErrorMessageCollection errorCollection)
464     {
465         MessageBox.Show(errorCollection.ToString(), "Errors occurred",
466             MessageBoxButtons.OK, MessageBoxIcon.Error);
467     }
468
469     /// <summary>
470     /// Asks fir confirmation of a deletion.
471     /// </summary>
472     /// <returns></returns>
473     public bool ConfirmDelete()
474     {
475         DialogResult result = MessageBox.Show("Are you sure you want to delete
476             this item?", "Confirm delete", MessageBoxButtons.YesNo, MessageBoxIcon
477             .Question);
478         return result == DialogResult.Yes;
479     }
480
481     /// <summary>
482     /// Asks fir confirmation of closing the application.
483     /// </summary>
484     /// <returns></returns>
485     public bool ConfirmClose()
486     {
487         DialogResult result = MessageBox.Show("Are you sure you want to close the
488             application?", "Confirm close", MessageBoxButtons.YesNo,
489             MessageBoxIcon.Question);
490         return result == DialogResult.Yes;
491     }
492
493     #endregion

```

```

489
490 #region Private methods
491
492 /// <summary>
493 /// Switches the BankAccount Controls depending on the selection.
494 /// </summary>
495 /// <param name="enabled">True if controls shall be enabled, false otherwise
496   .</param>
497 private void SwitchBankAccountControls(bool enabled)
498 {
499     this.deleteBankAccountToolStripMenuItem.Enabled = enabled;
500     this.deleteBankAccountToolStripButton.Enabled = enabled;
501     this.accountNumberTextBox.Enabled = enabled;
502     this.nameTextBox.Enabled = enabled;
503     this.depositQuantityTextBox.Enabled = enabled;
504     this.withdrawQuantityTextBox.Enabled = enabled;
505 }
506
507 /// <summary>
508 /// Switches the StockItem Controls depending on the selection.
509 /// </summary>
510 /// <param name="enabled">True if controls shall be enabled, false otherwise
511   .</param>
512 private void SwitchStockItemControls(bool enabled)
513 {
514     this.deleteStockItemToolStripButton.Enabled = enabled;
515     this.deleteStockItemToolStripMenuItem.Enabled = enabled;
516     this.stockCodeTextBox.Enabled = enabled;
517     this.itemNameTextBox.Enabled = enabled;
518     this.supplierNameTextBox.Enabled = enabled;
519     this.reqStockTextBox.Enabled = enabled;
520     this.currStockTextBox.Enabled = enabled;
521     this.priceTextBox.Enabled = enabled;
522 }
523
524 private void DisplayError(Control form)
525 {
526     backgroundColorChanged.Add(form);
527     form.BackColor = Color.MistyRose;
528 }
529
530 private void LoadFilePathSettings()
531 {
532     Settings settings = Settings.Default;
533
534     String filePathStockItems = settings.StockItemFilePath;
535     String filePathBankAccounts = settings.BankAccountFilePath;
536     if (!String.IsNullOrEmpty(filePathStockItems))
537     {
538         this._Presenter.SetupStockItemFilePath(filePathStockItems);
539     }
540 }

```

```

538         if (!String.IsNullOrEmpty(filePathBankAccounts))
539         {
540             this._Presenter.SetupBankAccountsFilePath(filePathBankAccounts);
541         }
542     }
543
544     private void SetupDataBindings()
545     {
546         stockItemsListBox.DataSource = _Model.StockItems;
547         stockItemsListBox.DisplayMember = "Name";
548
549         /*
550          * The datasourceupdate mode is set to "Never".
551          * This leads to the ability to enforce the use of the presenter to update
552            the values in the model.
553          * This way the validation errors can be handled by the presenter thus
554            leading to better separation of concerns.
555          */
556         stockCodeTextBox.DataBindings.Add("Text", _Model.StockItems, "StockCode",
557             false, DataSourceUpdateMode.Never);
558         itemNameTextBox.DataBindings.Add("Text", _Model.StockItems, "Name", false,
559             DataSourceUpdateMode.Never);
560         supplierNameTextBox.DataBindings.Add("Text", _Model.StockItems, "
561             SupplierName", false, DataSourceUpdateMode.Never);
562         currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
563             ", false, DataSourceUpdateMode.Never);
564         reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
565             ", false, DataSourceUpdateMode.Never);
566         priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
567             , DataSourceUpdateMode.Never);
568
569         bankAccountsListBox.DataSource = _Model.BankAccounts;
570         bankAccountsListBox.DisplayMember = "AccountNumber";
571
572         accountNumberTextBox.DataBindings.Add("Text", _Model.BankAccounts, "
573             AccountNumber", false, DataSourceUpdateMode.Never);
574         nameTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Surname", false
575             , DataSourceUpdateMode.Never);
576         balanceTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Balance",
577             false, DataSourceUpdateMode.Never);
578     }
579
580     private void CheckQuantity()
581     {
582         String newText = quantityTextBox.Text;
583         int parseInt = 0;
584         if (int.TryParse(newText, out parseInt))
585         {
586             placeOrderButton.Enabled = true;
587         }
588     }

```

```

578         }
579         else
580         {
581             placeOrderButton.Enabled = false;
582         }
583     }
584
585     #endregion
586
587 }
588 }
589 }

```

Listing 6: MainWindow.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using System.Collections.Specialized;
10 using System.Configuration;
11 using Assessment_One.Properties;
12
13 namespace Assessment_One
14 {
15     public partial class SettingsWindow : Form
16     {
17         public SettingsWindow()
18         {
19             InitializeComponent();
20             SetUpTextBoxes();
21         }
22
23         private void SetUpTextBoxes()
24         {
25             Settings settings = Settings.Default;
26             stockItemsFilePathTextBox.Text = settings.StockItemFilePath;
27             bankAccountFilePathTextBox.Text = settings.BankAccountFilePath;
28         }
29
30         private void chooseStockItemsFilePath_Click(object sender, EventArgs e)
31         {
32             this.openFileDialog.ShowDialog();
33             OpenFileDialog file = this.openFileDialog;
34             this.stockItemsFilePathTextBox.Text = file.FileName;
35         }
36

```

```

37     private void chooseBankAccountsFilePath_Click(object sender, EventArgs e)
38     {
39         this.openFileDialog.ShowDialog();
40         OpenFileDialog file = this.openFileDialog;
41         this.bankAccountFilePathTextBox.Text = file.FileName;
42     }
43
44     private void cancelButton_Click(object sender, EventArgs e)
45     {
46         this.Dispose();
47     }
48
49     private void applyButton_Click(object sender, EventArgs e)
50     {
51         String stockItemsFilePath = stockItemsFilePathTextBox.Text;
52         String bankAccountsFilePath = bankAccountFilePathTextBox.Text;
53         this.SaveApplicationSettings(stockItemsFilePath, bankAccountsFilePath);
54         this.Dispose();
55     }
56
57     private void SaveApplicationSettings(string stockItemsFilePath, string
58         bankAccountsFilePath)
59     {
60         Settings settings = Settings.Default;
61         settings.StockItemFilePath = stockItemsFilePath;
62         settings.BankAccountFilePath = bankAccountsFilePath;
63         settings.Save();
64     }
65 }

```

Listing 7: Settings.cs

A.2 APPLICATION LOGIC

A.2.1 *Interfaces Package*

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Interfaces
7  {
8      /// <summary>
9      /// Utilized by the presenter to get the necessary values from a view.
10     /// </summary>
11     public interface IBankAccountView

```



```

12     {
13         int AccountNumber { get; }
14         String Surname { get; }
15         double Balance { get; }
16     }
17 }

```

Listing 8: IBankAccountView.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using ApplicationLogic.Model;
6
7  namespace ApplicationLogic.Interfaces
8  {
9      /// <summary>
10     /// Utilized by the presenter to get the necessary values from a view.
11     /// </summary>
12     public interface ICongregateView
13     {
14         StockItem StockItem { get; }
15
16         BankAccount BankAccount { get; }
17
18         int Quantity { get; }
19
20         double Deposit { get; }
21
22         double Withdraw { get; }
23
24         bool ConfirmDelete();
25
26         bool ConfirmClose();
27
28         void DisplayValidationErrors(ErrorMessageCollection errorCollection);
29     }
30 }

```

Listing 9: ICongregateView.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Interfaces
7  {
8      /// <summary>

```

```

9      /// Used to ensure all objects will be able to persited via FileHandler class.
10     /// </summary>
11     /// <typeparam name="T"></typeparam>
12     public interface ICSVSerializable<T>
13     {
14         String CsvRepresentation();
15
16         T ParseFromString(String stringRepresentation);
17     }
18 }

```

Listing 10: ICSVSerializable.cs

```

1  i»using System;
2  namespace ApplicationLogic.Interfaces
3  {
4      /// <summary>
5      /// Utilized by the presenter to get the necessary values from a view.
6      /// </summary>
7      public interface IStockItemView
8      {
9          int CurrentStock { get; }
10         string ItemName { get; }
11         int RequiredStock { get; }
12         string StockCode { get; }
13         string SupplierName { get; }
14         double UnitCost { get; }
15     }
16 }

```

Listing 11: IStockItemView.cs

```

1  using System;
2  using System.ComponentModel;
3  using ApplicationLogic.Model;
4
5  namespace ApplicationLogic.Interfaces
6  {
7      /// <summary>
8      /// Utilized by the View to set up the data binding to the lists in the model
9      /// </summary>
10     public interface IViewModel
11     {
12         BindingList<StockItem> StockItems { get; }
13         BindingList<BankAccount> BankAccounts { get; }
14     }
15 }

```

Listing 12: IViewModel.cs

A.2.2 Model Package

```

1 using System;
2 using System.ComponentModel;
3 using System.Security.Cryptography;
4 using ApplicationLogic.Interfaces;
5 using System.Collections;
6 using System.Collections.Generic;
7
8 namespace ApplicationLogic.Model
9 {
10     /// <summary>
11     /// Handles persistence issues and ensure the correct sequence of method calls.
12     /// </summary>
13     public class AppDataManager : IViewModel
14     {
15         private BindingList<StockItem> _StockItems;
16         /// <summary>
17         /// Gets or sets the stock items.
18         /// </summary>
19         /// <value>The stock items.</value>
20         public BindingList<StockItem> StockItems
21         {
22             get { return _StockItems; }
23             set { _StockItems = value; }
24         }
25
26         private BindingList<BankAccount> _BankAccounts;
27         /// <summary>
28         /// Gets or sets the bank accounts.
29         /// </summary>
30         /// <value>The bank accounts.</value>
31         public BindingList<BankAccount> BankAccounts
32         {
33             get { return _BankAccounts; }
34             set { _BankAccounts = value; }
35         }
36
37         /// <summary>
38         /// Gets or sets the stock item handler.
39         /// </summary>
40         /// <value>The stock item handler.</value>
41         public FileHandler<StockItem> StockItemHandler { get; private set; }
42         /// <summary>
43         /// Gets or sets the bank account handler.
44         /// </summary>
45         /// <value>The bank account handler.</value>
46         public FileHandler<BankAccount> BankAccountHandler { get; private set; }
47
48         public AppDataManager()
49         {

```

```

50         this.StockItems = new BindingList<StockItem>();
51         this.BankAccounts = new BindingList<BankAccount>();
52         this.StockItemHandler = new FileHandler<StockItem>();
53         this.BankAccountHandler = new FileHandler<BankAccount>();
54     }
55
56     /// <summary>
57     /// Creates a new StockItem and initializes it with dummy values.
58     /// Adds the item to the StockItem collection.
59     /// </summary>
60     public void CreateNewStockItem()
61     {
62         StockItem si = new StockItem("oooo", "Dummy Item", "None", 0.0, 0, 0);
63         this.StockItems.Add(si);
64     }
65
66     /// <summary>
67     /// Deletes a StockItem from the StockItem collection.
68     /// Throws an ArgumentException if the item can not be found.
69     /// </summary>
70     public void DeleteStockItem(StockItem si)
71     {
72         if (this.StockItems.Contains(si))
73         {
74             this.StockItems.Remove(si);
75         }
76         else
77         {
78             throw new ArgumentException("Item to delete not present.");
79         }
80     }
81
82     /// <summary>
83     /// Creates a new BankAccount and initializes it with dummy values.
84     /// Adds the item to the BankAccount collection.
85     /// </summary>
86     public void CreateNewBankAccount()
87     {
88         BankAccount ba = new BankAccount(0, "Dummy Account", 0.0);
89         this.BankAccounts.Add(ba);
90     }
91
92     /// <summary>
93     /// Deletes a BankAccount from the BankAccount collection.
94     /// Throws an ArgumentException if the account can not be found.
95     /// </summary>
96     public void DeleteBankAccount(BankAccount ba)
97     {
98         if (this.BankAccounts.Contains(ba))
99         {
100

```

```

101         this.BankAccounts.Remove(ba);
102     }
103     else
104     {
105         throw new ArgumentException("Item to delete not present.");
106     }
107 }
108
109 /// <summary>
110 /// Attempts to edit specified StockItem in the StockItem collection.
111 /// Throws an ArgumentException if the item can not be found.
112 /// </summary>
113 /// <param name="si">StockItem to be edited.</param>
114 /// <param name="stockCode">New StockCode</param>
115 /// <param name="supplier">New SupplierName</param>
116 /// <param name="name">New Name</param>
117 /// <param name="currentStock">New CurrentStock</param>
118 /// <param name="reqStock">New RequiredStock</param>
119 /// <param name="price">New Price</param>
120 internal void EditStockItem(StockItem si, string stockCode, string supplier,
    string name, int currentStock, int reqStock, double price)
121 {
122     if (si != null)
123     {
124         si.EditStockItem(stockCode, name, supplier, price, reqStock,
            currentStock);
125     }
126     else
127     {
128         throw new ArgumentNullException("Stock item to edit not present.");
129     }
130 }
131
132 /// <summary>
133 /// Attempts to edit specified BankAccount in the BankAccount collection.
134 /// Throws an ArgumentException if the account can not be found.
135 /// </summary>
136 /// <param name="ba">BankAccount to be edited</param>
137 /// <param name="surname">New surname</param>
138 /// <param name="accountNumber">New accountnumber</param>
139 internal void EditBankAccount(BankAccount ba, string surname, int
    accountNumber)
140 {
141     if (ba != null)
142     {
143         ba.EditBankAccount(surname, accountNumber);
144     }
145     else
146     {
147         throw new ArgumentNullException("Bank account to edit not present.");
148     }

```

```

149     }
150
151
152     /// <summary>
153     /// Validates the changes that may be made to a stock item.
154     /// </summary>
155     /// <param name="accountNumber">New account number</param>
156     /// <param name="surname">New surname</param>
157     /// <returns>True if new values are valid. False otherwise.</returns>
158     internal bool ValidateStockItem(string stockCode, string name, string supplier
159                                     , double price, int reqStock, int currentStock)
160     {
161         bool areValuesValid = StockItem.Validate(stockCode, name, supplier, price,
162                                                     reqStock, currentStock);
163         return areValuesValid;
164     }
165
166     /// <summary>
167     /// Stores the errors that occurred in the last validation of stock item data.
168     /// </summary>
169     /// <returns>The errors that occurred.</returns>
170     internal ErrorMessageCollection StockItemErrors()
171     {
172         return StockItem.ErrorMessages; ;
173     }
174
175     /// <summary>
176     /// Clears the errors of the last stock item validation.
177     /// </summary>
178     internal void ClearStockItemErrors()
179     {
180         StockItem.ErrorMessages.Clear();
181     }
182
183     /// <summary>
184     /// Clears the errors of the last bank account validation.
185     /// </summary>
186     internal void ClearBankAccountErrors()
187     {
188         BankAccount.ErrorMessages.Clear();
189     }
190
191     /// <summary>
192     /// Stores the errors that occurred in the last validation of bank account data
193     .
194     /// </summary>
195     /// <returns>The errors that occurred.</returns>
196     internal ErrorMessageCollection BankAccountErrors()
197     {
198         return BankAccount.ErrorMessages;
199     }

```

```

197     }
198
199     /// <summary>
200     /// Validates the changes that may be made to a bank account.
201     /// </summary>
202     /// <param name="accountNumber">New account number</param>
203     /// <param name="surname">New surname</param>
204     /// <returns>True if new values are valid. False otherwise.</returns>
205     internal bool ValidateBankAccount(int accountNumber, string surname)
206     {
207         bool areValidValues = BankAccount.Validate(accountNumber, surname);
208         return areValidValues;
209     }
210
211     /// <summary>
212     /// Attempts to order an item. If no quantity was provided the required
213     /// quantity will be ordered.
214     /// Otherwise the provided quantity will be ordered.
215     /// </summary>
216     /// <param name="indexStockItem">Index of the stock item in the stock items
217     /// list.</param>
218     /// <param name="indexBankAccount">Index of the bank account in the bank
219     /// account list.</param>
220     /// <param name="quantity">The quantity to be ordered. 0 orders the required
221     /// quantity.</param>
222     public void OrderItem(StockItem si, BankAccount ba, int quantity)
223     {
224         if (ba != null && si != null)
225         {
226             bool buyExcessStock = false;
227             if (quantity == 0)
228             {
229                 quantity = si.RequiredStock;
230             }
231             else
232             {
233                 /*
234                  * It is possible to order more than the needed quantity.
235                  */
236                 buyExcessStock = true;
237             }
238             double priceOfOrder = quantity * si.UnitCost;
239             if (priceOfOrder <= ba.Balance)
240             {
241                 ba.Transfer(1, priceOfOrder);
242                 si.CurrentStock += quantity;
243                 /*
244                  * Allow the user to buy more than needed.
245                  */
246                 if (buyExcessStock)
247                 {

```

```

244         if (quantity > si.RequiredStock)
245         {
246             si.RequiredStock = 0;
247         }
248     }
249 }
250 else
251 {
252     throw new NotEnoughFundsException(String.Format("Not enough funds
        on bank account {0} to place an order for {1} £", ba.
        AccountNumber, priceOfOrder));
253 }
254 }
255 else
256 {
257     throw new ArgumentNullException("Stock item or bank account provided
        do not exist.");
258 }
259 }
260
261 /// <summary>
262 /// Attempts to deposit the requested amount from the specified bank account.
263 /// </summary>
264 /// <param name="indexBankAccount"></param>
265 /// <param name="amount"></param>
266 internal void Deposit(BankAccount ba, double amount)
267 {
268     if (ba != null)
269     {
270         ba.Deposit(amount);
271     }
272     else
273     {
274         throw new ArgumentNullException("Provided bank account does not exist.
            ");
275     }
276 }
277
278 /// <summary>
279 /// Attempts to withdraw the specified amount from the specified bank account.
280 /// </summary>
281 /// <param name="indexBankAccount"></param>
282 /// <param name="amount"></param>
283 internal void Withdraw(BankAccount ba, double amount)
284 {
285     if (ba != null)
286     {
287         ba.Withdraw(amount);
288     }
289     else
290     {

```



```

291         throw new ArgumentNullException("Bank account provided does not exist.
292             ");
293     }
294 }
295
296 /// <summary>
297 /// Will reload the StockItem collection from the specified file.
298 /// Will overwrite the currently existing StockItem collection.
299 /// </summary>
300 /// <param name="filePath"></param>
301 internal void LoadStockItemsFromFile(string filePath)
302 {
303     this.StockItemHandler.ReadFilePath = filePath;
304     IList<StockItem> stockItems = StockItemHandler.LoadFromFile(new StockItem
305         ());
306     this.StockItems.Clear();
307     foreach (StockItem item in stockItems)
308     {
309         this.StockItems.Add(item);
310     }
311 }
312
313 /// <summary>
314 /// Will reload the BankAccount collection from the specified file.
315 /// Will overwrite the currently existing BankAccount collection.
316 /// </summary>
317 /// <param name="filePath">The path to the file</param>
318 internal void LoadBankAccountsFromFile(string filePath)
319 {
320     this.BankAccountHandler.ReadFilePath = filePath;
321     IList<BankAccount> bankAccounts = BankAccountHandler.LoadFromFile(new
322         BankAccount());
323     this.BankAccounts.Clear();
324     foreach (BankAccount item in bankAccounts)
325     {
326         this.BankAccounts.Add(item);
327     }
328 }
329
330 /// <summary>
331 /// Will save the current StockItem collection to the specified file.
332 /// </summary>
333 /// <param name="filePath">The path to the file</param>
334 internal void SaveStockItemsToFile(string filePath)
335 {
336     this.StockItemHandler.WriteFilePath = filePath;
337     this.StockItemHandler.SaveToFile(this.StockItems);
338 }
339
340 /// <summary>
341 /// Will save the current BankAccount collection to the specified file.

```

```

339     /// </summary>
340     /// <param name="filePath">The path to the file</param>
341     internal void SaveBankAccountsToFile(string filePath)
342     {
343         this.BankAccountHandler.WriteFilePath = filePath;
344         this.BankAccountHandler.SaveToFile(this.BankAccounts);
345     }
346
347     /// <summary>
348     /// Will save the current BankAccount collection to the file stored in the
349     /// FileHandler.
350     /// Throws NoFilePathSetException if no path is set.
351     /// </summary>
352     internal void SaveBankAccountsToFile()
353     {
354         this.BankAccountHandler.SaveToFile(this.BankAccounts);
355     }
356
357     /// <summary>
358     /// Will save the current BankAccount collection to the file stored in the
359     /// FileHandler.
360     /// Throws NoFilePathSetException if no path is set.
361     /// </summary>
362     internal void SaveStockItemsToFile()
363     {
364         this.StockItemHandler.SaveToFile(this.StockItems);
365     }
366 }

```

Listing 13: AppDataManager.cs

```

1  i>using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.ComponentModel;
6  using ApplicationLogic.Interfaces;
7
8  namespace ApplicationLogic.Model
9  {
10     /// <summary>
11     /// Pseudo bank account to allow the placement of an order.
12     /// </summary>
13     public class BankAccount : INotifyPropertyChanged, ICSVSerializable<
14         BankAccount>
15     {
16         private int _AccountNumber;
17         /// <summary>
18         /// Gets or sets the account number.
19         /// </summary>

```

```

19     /// <value>The account number.</value>
20     public virtual int AccountNumber {
21         get { return _AccountNumber; }
22         set {
23             _AccountNumber = value;
24             this.NotifyPropertyChanged ("AccountNumber");
25         }
26     }
27
28     private String _Surname;
29     /// <summary>
30     /// Gets or sets the surname.
31     /// </summary>
32     /// <value>The surname.</value>
33     public virtual String Surname {
34         get { return _Surname; }
35         set {
36             _Surname = value;
37             this.NotifyPropertyChanged ("Surname");
38         }
39     }
40
41     private double _Balance;
42     /// <summary>
43     /// Gets or sets the balance.
44     /// </summary>
45     /// <value>The balance.</value>
46     public virtual double Balance {
47         get { return _Balance; }
48         private set {
49             if (value < 0.0) {
50                 throw new ArgumentException ("This class does
51                 not allow a balance smaller than 0.");
52             } else {
53                 _Balance = value;
54                 this.NotifyPropertyChanged ("Balance");
55             }
56         }
57     }
58
59     public static ErrorMessageCollection ErrorMessages = new
60         ErrorMessageCollection ();
61
62     /// <summary>
63     /// Initializes a new instance of the <see cref="BankAccount"/> class.
64     /// </summary>
65     public BankAccount ()
66     {
67

```

```

68      /// <summary>
69      /// Initializes a new instance of the <see cref="BankAccount"/> class.
70      /// </summary>
71      /// <param name="acc">The acc.</param>
72      /// <param name="name">The name.</param>
73      /// <param name="balance">The balance.</param>
74      public BankAccount (int acc, string name, double balance)
75      {
76          this.AccountNumber = acc;
77          this.Surname = name;
78          this.Balance = balance;
79      }
80
81      /// <summary>
82      /// Allows the withdrawal of money from this account.
83      /// Credit is not granted.
84      /// </summary>
85      /// <param name="amount">Amount to be withdrawn - must be greater than
86          0.</param>
87      public void Withdraw (double amount)
88      {
89          if (amount > 0.0) {
90              if (Balance >= amount) {
91                  this.Balance -= amount;
92              } else {
93                  throw new NotEnoughFundsException ("Not enough
94                      founds on bank account to withdraw.");
95              }
96          } else {
97              throw new ArgumentException("To deposit money please use the
98                  appropriate function.");
99          }
100      }
101
102      /// <summary>
103      /// Allows the deposit of money to this account.
104      /// </summary>
105      /// <param name="amount">Amount to be deposited - must be greater than
106          0.</param>
107      public void Deposit (double amount)
108      {
109          if (amount <= 0.0) {
110              throw new ArgumentException ("To withdraw money please
111                  use the appropriate function.");
112          } else {
113              this.Balance += amount;
114          }
115      }
116
117      /// <summary>
118      /// Amount to be transfered to another account.

```

```

114     /// NOTE: This method is a fake to simulate "real" banking. The money
115     /// will not be transfered to any account.
116     /// </summary>
117     /// <param name="amount">Amount to be transfered - must be greater
118     /// than 0.</param>
119     /// <param name="accountNumber">Account number to transfer the money
120     /// to.</param>
121     public void Transfer (int accountNumber, double amount)
122     {
123         if (amount >= 0.0) {
124             if (this.Balance >= amount) {
125                 this.Balance -= amount;
126                 // TODO: In reality: fancy logic to transfer
127                 // money.
128             } else {
129                 throw new NotEnoughFundsException ("There are
130                 not enough funds present to fulfill the
131                 required action.");
132             }
133         } else {
134             throw new ArgumentException ("It is not possible to
135             transfer funds from another account to yours.");
136         }
137     }
138
139     /// <summary>
140     /// Validates a set of possible changes to a BankAccount.
141     /// </summary>
142     /// <param name="accountNumber">AccountNumber to be verified</param>
143     /// <param name="surname">Surname to be verified</param>
144     /// <returns>True if the values would be valid, false otherwise.</
145     /// returns>
146     static internal bool Validate (int accountNumber, String surname)
147     {
148         if (String.IsNullOrEmpty (surname)) {
149             ErrorMessage.Add (new ErrorMessage ("Need the name of
150             the account owner."));
151         }
152         if (accountNumber <= 0) {
153             ErrorMessage.Add (new ErrorMessage ("Need a valid
154             account number: greater o."));
155         }
156         return ErrorMessage.Count == 0;
157     }
158
159     public event PropertyChangedEventHandler PropertyChanged;
160
161     private void NotifyPropertyChanged (String info)
162     {
163         if (PropertyChanged != null) {

```

```

154         PropertyChanged (this, new PropertyChangedEventArgs (
155             info));
156     }
157
158     internal void EditBankAccount (string surname, int accountNumber)
159     {
160         this.Surname = surname;
161         this.AccountNumber = accountNumber;
162     }
163
164     /// <summary>
165     /// Returns the current BankAccount object as a CSV-String.
166     /// </summary>
167     /// <returns>Representation of the current object as CSV-String.</
168     returns>
169     public string CsvRepresentation ()
170     {
171         return String.Format ("{0},{1},{2}", this.AccountNumber, this.
172             Surname, this.Balance);
173     }
174
175     /// <summary>
176     /// Attempts to create a BankAccount object from a string.
177     /// </summary>
178     /// <param name="stringRepresentation">The String to be parsed to bank
179     account.</param>
180     /// <returns>BankAccount object.</returns>
181     public BankAccount ParseFromString (string stringRepresentation)
182     {
183         string[] split = stringRepresentation.Split (',');
184         if (split.Length == 3)
185         {
186             String accountNumber = split[0];
187             String surname = split[1];
188             String balance = split[2];
189             int accNumber = 0;
190             double bal = 0;
191             if (!String.IsNullOrEmpty(accountNumber))
192             {
193                 accNumber = int.Parse(accountNumber);
194             }
195             if (!String.IsNullOrEmpty(balance))
196             {
197                 bal = double.Parse(balance);
198             }
199             return new BankAccount(accNumber, surname, bal);
200         }
201         else
202         {

```

```

200         throw new WrongStringToParseException("Can not parse a bank account
201         }
202         }
203     }
204 }

```

Listing 14: BankAccount.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Model
7  {
8      public class ErrorMessage
9      {
10         public String Message { get; set; }
11         public String Source { get; set; }
12
13         public ErrorMessage(string p)
14         {
15             this.Message = p;
16         }
17
18         public override string ToString()
19         {
20             return Message;
21         }
22     }
23 }

```

Listing 15: ErrorMessage.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Model
7  {
8      /// <summary>
9      /// Used to store meaningful error messages for further use.
10     /// </summary>
11     public class ErrorMessageCollection : List<ErrorMessage>
12     {
13         public override string ToString()
14         {
15             StringBuilder sb = new StringBuilder();

```

```

16
17     foreach (ErrorMessage item in this)
18     {
19         if (sb.Length > 0)
20         {
21             sb.Append(Environment.NewLine);
22         }
23
24         sb.Append(item.ToString());
25     }
26
27     return sb.ToString();
28 }
29 }
30 }

```

Listing 16: ErrorMessageCollection.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using ApplicationLogic.Interfaces;
5  using System.IO;
6  using ApplicationLogic.Model;
7
8  namespace ApplicationLogic
9  {
10     /// <summary>
11     /// Handles reading and writing from files.
12     /// </summary>
13     public class FileHandler<T> where T : ICSVSerializable<T>
14     {
15
16         public String ReadFilePath;
17         public String WriteFilePath;
18
19         public FileHandler()
20         {
21         }
22
23         public FileHandler(String readFilePath, String writeFilePath)
24         {
25             this.ReadFilePath = readFilePath;
26             this.WriteFilePath = writeFilePath;
27         }
28
29         /// <summary>
30         /// Attempts to write the specified collection to the file specified in the
31         /// Handler.
32         /// Throws NoFilePathSetException if no file has yet been set.
33         /// </summary>

```



```

33  /// <param name="elements">Collection to be saved to file.</param>
34  public void SaveToFile(IList<T> elements)
35  {
36      if (!String.IsNullOrEmpty(this.WriteFilePath))
37      {
38          FileStream writeFile = File.Open(WriteFilePath, FileMode.Create);
39          using (StreamWriter sw = new StreamWriter(writeFile))
40          {
41              foreach (T item in elements)
42              {
43                  sw.Write(item.CsvRepresentation());
44                  sw.Write("\n");
45              }
46          }
47      }
48      else
49      {
50          throw new NoFilePathSetException("No file path to write to set.");
51      }
52  }
53
54
55  /// <summary>
56  /// Attempts to read a collection of items from the specified file.
57  /// Throws NoFilePathSetException if no file has yet been set.
58  /// </summary>
59  /// <param name="item">Parameter needed to construct the objects.</param>
60  /// <returns>Collection of items.</returns>
61  public IList<T> LoadFromFile(T item)
62  {
63      if (!String.IsNullOrEmpty(this.ReadFilePath))
64      {
65          FileStream readFile = File.Open(ReadFilePath, FileMode.Open);
66          List<T> returnList = new List<T>();
67          using (StreamReader sr = new StreamReader(readFile))
68          {
69              String readString = "";
70              while ((readString = sr.ReadLine()) != null)
71              {
72                  try {
73                      T t = item.ParseFromString(
74                          readString);
75                      returnList.Add(t);
76                  } catch (FormatException ex) {
77                      Console.WriteLine(ex.
78                          StackTrace);
79                  }
80              }
81              return returnList;
82          }
83      }
84  }

```

```

82         else
83         {
84             throw new NoFilePathSetException("No file path to write to set.");
85         }
86     }
87 }
88 }

```

Listing 17: FileHandler.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Model
7  {
8      public class NoFilePathSetException : Exception
9      {
10
11         public NoFilePathSetException(String msg)
12             : base(msg)
13         {
14         }
15     }
16 }

```

Listing 18: NoFilePathSetException.cs

```

1  i»using System;
2
3  namespace ApplicationLogic.Model
4  {
5      /// <summary>
6      /// Description of NotEnoughFundsException.
7      /// </summary>
8      public class NotEnoughFundsException: Exception
9      {
10         public NotEnoughFundsException(String message): base(message)
11         {
12         }
13     }
14 }

```

Listing 19: NotEnoughFundsException.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;

```

```

4 using System.Text;
5 using System.Text.RegularExpressions;
6 using System.ComponentModel;
7 using ApplicationLogic.Interfaces;
8
9 namespace ApplicationLogic.Model
10 {
11     /// <summary>
12     /// Stores all necessary data for a StockItem.
13     /// </summary>
14     public class StockItem : INotifyPropertyChanged, ICSVSerializable<StockItem>
15     {
16         private const string REGEX = "^[0-9]{4}$";
17         private String _StockCode;
18         public virtual String StockCode {
19             get { return _StockCode; }
20             set {
21                 if (!IsValidStockCode (value)) {
22                     throw new ArgumentException ("Provided
23                                     stockcode did not match designated format.
24                                     ");
25                 } else {
26                     this._StockCode = value;
27                     this.NotifyPropertyChanged ("StockCode");
28                 }
29             }
30         }
31         private String _Name;
32         public virtual String Name {
33             get { return _Name; }
34             set {
35                 _Name = value;
36                 this.NotifyPropertyChanged ("Name");
37             }
38         }
39         private String _SupplierName;
40         public virtual String SupplierName {
41             get { return _SupplierName; }
42             set {
43                 _SupplierName = value;
44                 this.NotifyPropertyChanged ("SupplierName");
45             }
46         }
47         private double _UnitCost;
48         public virtual double UnitCost {
49             get { return _UnitCost; }
50             private set {
51                 if (value < 0.0) {
52

```

```

53         throw new ArgumentException ("Price can not be
54             lower than 0.");
55     } else {
56         _UnitCost = value;
57         this.NotifyPropertyChanged ("UnitCost");
58     }
59 }
60
61
62 private int _RequiredStock;
63 public virtual int RequiredStock {
64     get { return _RequiredStock; }
65     set {
66         if (value < 0) {
67             throw new ArgumentException ("Can not require
68                 less than 0 items.");
69         } else {
70             _RequiredStock = value;
71             this.NotifyPropertyChanged ("RequiredStock");
72         }
73     }
74 }
75
76 private int _CurrentStock;
77 public virtual int CurrentStock {
78     get { return _CurrentStock; }
79     set {
80         if (value < 0) {
81             throw new ArgumentException ("Current stock
82                 can not be less than 0 items.");
83         } else {
84             _CurrentStock = value;
85             this.NotifyPropertyChanged ("CurrentStock");
86         }
87     }
88 }
89
90 public static ErrorMessageCollection ErrorMessages = new
91     ErrorMessageCollection ();
92
93 public StockItem ()
94 {
95 }
96
97 public StockItem (String stockCode, String name, String supplierName,
98     double unitCost, int required, int currentStock)
99 {
100     this.StockCode = stockCode;

```

```

99         this.Name = name;
100         this.SupplierName = supplierName;
101         this.UnitCost = unitCost;
102         this.RequiredStock = required;
103         this.CurrentStock = currentStock;
104     }
105
106     /// <summary>
107     /// Checks if the stock code conforms to a certain format.
108     /// Current format is exactly four numbers, with leading 0 allowed.
109     /// </summary>
110     /// <param name="value">The string that must be checked against the
111     /// schema.</param>
112     /// <returns>True if the string conforms to the schema, false
113     /// otherwise.</returns>
114     public static bool IsValidStockCode (string value)
115     {
116         if (String.IsNullOrEmpty (value)) {
117             throw new ArgumentNullException ("Provided stockcode
118             was null or empty.");
119         } else {
120             // TODO: Do not use magic numbers in code
121             Regex regexp = new Regex (REGEX);
122             return regexp.IsMatch (value);
123         }
124     }
125
126     public void EditStockItem (String stockCode, String name, String
127     supplierName, double unitCost, int required, int currentStock)
128     {
129         this.StockCode = stockCode;
130         this.Name = name;
131         this.SupplierName = supplierName;
132         this.UnitCost = unitCost;
133         this.RequiredStock = required;
134         this.CurrentStock = currentStock;
135     }
136
137     /// <summary>
138     /// Validates a set of possible changes to a StockItem.
139     /// </summary>
140     /// <param name="stockCode">StockCodew to be verified</param>
141     /// <param name="name">Name to be verified</param>
142     /// <param name="supplierName">SupplierName to be verified</param>
143     /// <param name="unitCost">UnitCost to be verified</param>
144     /// <param name="required">RequiredStock to be verified</param>
145     /// <param name="currentStock">CurrentStock to be verified</param>
146     /// <returns>True if the values would be valid, false otherwise.</
147     returns>

```

```

144 public static bool Validate (String stockCode, String name, String
      supplierName, double unitCost, int required, int currentStock)
145 {
146     if (String.IsNullOrEmpty (stockCode) || !IsValidStockCode (
      stockCode)) {
147         ErrorMessage.Add (new ErrorMessage ("Need a stockcode
      that adheres to the stockcode format: 4 numbers."
      ));
148     }
149     if (String.IsNullOrEmpty (name)) {
150         ErrorMessage.Add (new ErrorMessage ("Need an item
      name."));
151     }
152     if (String.IsNullOrEmpty (supplierName)) {
153         ErrorMessage.Add (new ErrorMessage ("Need a supplier
      name."));
154     }
155     if (unitCost < 0.0) {
156         ErrorMessage.Add (new ErrorMessage ("Unit costs must
      be greater or equal 0."));
157     }
158     if (required < 0) {
159         ErrorMessage.Add (new ErrorMessage ("Required stock
      must be greater or equal 0."));
160     }
161     if (currentStock < 0) {
162         ErrorMessage.Add (new ErrorMessage ("Current stock
      must be greater or equal 0."));
163     }
164     return ErrorMessage.Count == 0;
165 }
166
167 public event PropertyChangedEventHandler PropertyChanged;
168
169 private void NotifyPropertyChanged (String info)
170 {
171     if (PropertyChanged != null) {
172         PropertyChanged (this, new PropertyChangedEventArgs (
      info));
173     }
174 }
175
176 /// <summary>
177 /// Returns the current StockItem object as a CSV-String.
178 /// </summary>
179 /// <returns>Representation of the current object as CSV-String.</
      returns>
180 public String CsvRepresentation ()
181 {

```

```

182         return String.Format ("{0},{1},{2},{3},{4},{5}", this.
            StockCode, this.Name, this.SupplierName, this.UnitCost,
            this.RequiredStock, this.CurrentStock);
183     }
184
185     /// <summary>
186     /// Attempts to create a StockItem object from a string.
187     /// </summary>
188     /// <param name="stringRepresentation">The String to be parsed to
        StockItem.</param>
189     /// <returns>StockItem object.</returns>
190     public StockItem ParseFromString (string stringRepresentation)
191     {
192         string[] split = stringRepresentation.Split (',');
193     if (split.Length == 6)
194     {
195         String stockCode = split[0];
196         String name = split[1];
197         String supplierName = split[2];
198         String unitCost = split[3];
199         String requiredStock = split[4];
200         String currentStock = split[5];
201         double cost = 0;
202         int reqStock = 0;
203         int currStock = 0;
204         if (!String.IsNullOrEmpty(unitCost))
205         {
206             cost = double.Parse(unitCost);
207         }
208         if (!String.IsNullOrEmpty(requiredStock))
209         {
210             reqStock = int.Parse(requiredStock);
211         }
212         if (!String.IsNullOrEmpty(currentStock))
213         {
214             currStock = int.Parse(currentStock);
215         }
216         return new StockItem(stockCode, name, supplierName, cost, reqStock,
            currStock);
217     }
218     else
219     {
220         throw new WrongStringToParseException("Can not parse a stock item from
            the provided string.");
221     }
222 }
223
224 }
225 }

```

Listing 20: StockItem.cs

A.2.3 *Presenter Package*

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Linq;
5  using System.Text;
6  using ApplicationLogic.Interfaces;
7  using ApplicationLogic.Model;
8
9  namespace ApplicationLogic.Presenter
10 {
11     /// <summary>
12     /// Presenter for the MainWindow: handles events and GUI-related part of the
13     /// application logic.
14     /// </summary>
15     public class CongregatePresenter
16     {
17         public ICongregateView _View;
18         public IStockItemView _StockItemView;
19         public IBankAccountView _BankAccountView;
20         public AppDataManager _Model;
21
22         /// <summary>
23         /// Initializes a new instance of the <see cref="CongregatePresenter"/> class.
24         /// </summary>
25         /// <param name="view">The view.</param>
26         /// <param name="stockItemView">The stock item view.</param>
27         /// <param name="bankAccountView">The bank account view.</param>
28         /// <param name="model">The model.</param>
29         public CongregatePresenter(ICongregateView view, IStockItemView stockItemView,
30             IBankAccountView bankAccountView, IViewModel model)
31         {
32             this._View = view;
33             this._StockItemView = stockItemView;
34             this._BankAccountView = bankAccountView;
35             this._Model = model as AppDataManager;
36         }
37
38         /// <summary>
39         /// <see cref="ApplicationLogic.Model.AppDataManagerClass"/>
40         /// </summary>
41         public void CreateNewStockItem()
42         {
43             this._Model.CreateNewStockItem();
44         }
45
46         /// <summary>
47         /// Deletes the stock item.
48         /// </summary>

```



```

48 public void DeleteStockItem()
49 {
50     if (this._View.ConfirmDelete())
51     {
52         StockItem si = this._View.StockItem;
53         this._Model.DeleteStockItem(si);
54     }
55 }
56
57 /// <summary>
58 /// Creates the new bank account.
59 /// </summary>
60 public void CreateNewBankAccount()
61 {
62     this._Model.CreateNewBankAccount();
63 }
64
65 /// <summary>
66 /// Deletes the bank account.
67 /// </summary>
68 public void DeleteBankAccount()
69 {
70     if (this._View.ConfirmDelete())
71     {
72         BankAccount ba = this._View.BankAccount;
73         this._Model.DeleteBankAccount(ba);
74     }
75 }
76
77 /// <summary>
78 /// Edits the stock item.
79 /// </summary>
80 public void EditStockItem()
81 {
82     try
83     {
84         StockItem si = this._View.StockItem;
85         String stockCode = this._StockItemView.StockCode;
86         String supplier = this._StockItemView.SupplierName;
87         String name = this._StockItemView.ItemName;
88         int currentStock = this._StockItemView.CurrentStock;
89         int reqStock = this._StockItemView.RequiredStock;
90         double price = this._StockItemView.UnitCost;
91         bool areValuesValid = this._Model.ValidateStockItem(stockCode, name,
92             supplier, price, reqStock, currentStock);
93         if (areValuesValid)
94         {
95             this._Model.EditStockItem(si, stockCode, supplier, name,
96                 currentStock, reqStock, price);
97         }
98     }
99     catch { }
100 }

```

```

97         {
98             this._View.DisplayValidationErrors(this._Model.StockItemErrors());
99             this._Model.ClearStockItemErrors();
100         }
101     }
102     catch (FormatException e)
103     {
104         DisplayError(e);
105     }
106 }
107
108
109 /// <summary>
110 /// Edits the bank account.
111 /// </summary>
112 public void EditBankAccount()
113 {
114     try
115     {
116         BankAccount ba = this._View.BankAccount;
117         String surname = this._BankAccountView.Surname;
118         int accountNumber = this._BankAccountView.AccountNumber;
119         bool areValuesValid = this._Model.ValidateBankAccount(accountNumber,
120             surname);
121         if (areValuesValid)
122         {
123             this._Model.EditBankAccount(ba, surname, accountNumber);
124         }
125         else
126         {
127             this._View.DisplayValidationErrors(this._Model.BankAccountErrors());
128             this._Model.ClearBankAccountErrors();
129         }
130     }
131     catch (FormatException e)
132     {
133         DisplayError(e);
134     }
135 }
136
137 /// <summary>
138 /// Orders the item.
139 /// </summary>
140 public void OrderItem()
141 {
142     try
143     {
144         BankAccount ba = this._View.BankAccount;
145         StockItem si = this._View.StockItem;

```

```

146         this.EditStockItem();
147         int quantity = this._View.Quantity;
148         this._Model.OrderItem(si, ba, quantity);
149     }
150     catch (FormatException e)
151     {
152         DisplayError(e);
153     }
154     catch (NotEnoughFundsException e)
155     {
156         DisplayError(e);
157     }
158     catch (ArgumentNullException e)
159     {
160         DisplayError(e);
161     }
162 }
163
164 /// <summary>
165 /// Deposits this instance.
166 /// </summary>
167 public void Deposit()
168 {
169     try
170     {
171         BankAccount ba = this._View.BankAccount;
172         double amount = this._View.Deposit;
173         this._Model.Deposit(ba, amount);
174     }
175     catch (ArgumentNullException e)
176     {
177         DisplayError(e);
178     }
179     catch (ArgumentException e)
180     {
181         DisplayError(e);
182     }
183     catch (FormatException e)
184     {
185         DisplayError(e);
186     }
187     catch (NotEnoughFundsException e)
188     {
189         DisplayError(e);
190     }
191 }
192
193 /// <summary>
194 /// Withdraws this instance.
195 /// </summary>
196 public void Withdraw()

```

```

197     {
198         try
199         {
200             BankAccount ba = this._View.BankAccount;
201             double amount = this._View.Withdraw;
202             this._Model.Withdraw(ba, amount);
203         }
204         catch (ArgumentNullException e)
205         {
206             DisplayError(e);
207         }
208         catch (ArgumentException e)
209         {
210             DisplayError(e);
211         }
212         catch (FormatException e)
213         {
214             DisplayError(e);
215         }
216         catch (NotEnoughFundsException e)
217         {
218             DisplayError(e);
219         }
220     }
221
222     /// <summary>
223     /// Displays the error.
224     /// </summary>
225     /// <param name="e">The e.</param>
226     private void DisplayError(Exception e)
227     {
228         ErrorMessageCollection col = new ErrorMessageCollection();
229         col.Add(new ErrorMessage(e.Message));
230         this._View.DisplayValidationErrors(col);
231     }
232
233     /// <summary>
234     /// Closes the application.
235     /// </summary>
236     public void CloseApplication()
237     {
238         if (this._View.ConfirmClose())
239             Environment.Exit(1);
240     }
241
242     /// <summary>
243     /// Loads the stock items from file.
244     /// </summary>
245     /// <param name="filePath">The file path.</param>
246     public void LoadStockItemsFromFile(String filePath)
247     {

```

```

248         try
249         {
250             this._Model.LoadStockItemsFromFile(filePath);
251         }
252         catch (WrongStringToParseException e)
253         {
254             DisplayError(e);
255         }
256     }
257
258
259     /// <summary>
260     /// Loads the bank accounts from file.
261     /// </summary>
262     /// <param name="filePath">The file path.</param>
263     public void LoadBankAccountsFromFile(String filePath)
264     {
265         try
266         {
267             this._Model.LoadBankAccountsFromFile(filePath);
268         }
269         catch (WrongStringToParseException e)
270         {
271             DisplayError(e);
272         }
273     }
274
275     /// <summary>
276     /// Saves the stock items to file.
277     /// </summary>
278     /// <param name="filePath">The file path.</param>
279     public void SaveStockItemsToFile(String filePath)
280     {
281         this._Model.SaveStockItemsToFile(filePath);
282     }
283
284     /// <summary>
285     /// Saves the bank accounts to file.
286     /// </summary>
287     /// <param name="filePath">The file path.</param>
288     public void SaveBankAccountsToFile(String filePath)
289     {
290         this._Model.SaveBankAccountsToFile(filePath);
291     }
292
293     /// <summary>
294     /// Sets up stock item file path.
295     /// </summary>
296     /// <param name="filePathStockItems">The file path stock items.</param>
297     public void SetUpStockItemFilePath(string filePathStockItems)
298     {

```

```

299         // TODO: Check if good.
300         this._Model.StockItemHandler.ReadFilePath = filePathStockItems;
301         this._Model.StockItemHandler.WriteFilePath = filePathStockItems;
302         try
303         {
304             this._Model.LoadStockItemsFromFile(filePathStockItems);
305         }
306         catch (WrongStringToParseException e)
307         {
308             DisplayError(e);
309         }
310     }
311 }
312
313 /// <summary>
314 /// Sets up bank accounts file path.
315 /// </summary>
316 /// <param name="filePathBankAccounts">The file path bank accounts.</param>
317 public void SetUpBankAccountsFilePath(string filePathBankAccounts)
318 {
319     // TODO: Check if good.
320     this._Model.BankAccountHandler.ReadFilePath = filePathBankAccounts;
321     this._Model.BankAccountHandler.WriteFilePath = filePathBankAccounts;
322     try
323     {
324         this._Model.LoadBankAccountsFromFile(filePathBankAccounts);
325     }
326     catch (WrongStringToParseException e)
327     {
328         DisplayError(e);
329     }
330 }
331
332 /// <summary>
333 /// Saves the bank accounts to file.
334 /// </summary>
335 public void SaveBankAccountsToFile()
336 {
337     try
338     {
339         {
340             this._Model.SaveBankAccountsToFile();
341         }
342         catch (NoFilePathSetException e)
343         {
344             DisplayError(e);
345         }
346     }
347 }
348
349 /// <summary>
350 /// Saves the stock items to file.

```

```

350     /// </summary>
351     public void SaveStockItemsToFile()
352     {
353         try
354         {
355             this._Model.SaveStockItemsToFile();
356         }
357         catch (NoFilePathSetException e)
358         {
359             DisplayError(e);
360         }
361     }
362 }
363 }

```

Listing 21: CongregatePresenter.cs

A.3 TESTS

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using ApplicationLogic.Model;
6  using NUnit.Framework;
7
8  namespace NUnit_Tests.ApplicationLogic
9  {
10     [TestFixture]
11     public class BankAccountTest
12     {
13         private BankAccount ba;
14
15         [SetUp]
16         public void Setup()
17         {
18             ba = new BankAccount(123, "Test", 0.0);
19         }
20
21         [Test]
22         [ExpectedException(typeof(ArgumentException))]
23         public void TestBalanceUnder0()
24         {
25             ba = new BankAccount(123, "Test", -1.0);
26         }
27
28         [Test]
29         [ExpectedException(typeof(NotEnoughFundsException))]
30         public void TestWithdrawalWithTooHighValues()

```

```

31     {
32         double amountToWithdrawTooHigh = 50.0;
33         ba.Withdraw(amountToWithdrawTooHigh);
34     }
35
36     [Test]
37     [ExpectedException(typeof(ArgumentException))]
38     public void TestWithdrawalWithTooSmallValues()
39     {
40         double amountToWithdrawTooSmall = -10.0;
41         ba.Withdraw(amountToWithdrawTooSmall);
42     }
43
44     [Test]
45     [ExpectedException(typeof(ArgumentException))]
46     public void TestDepositWithTooSmallValue()
47     {
48         double amountToDepositTooSmall = -10;
49         ba.Deposit(amountToDepositTooSmall);
50     }
51
52     [Test]
53     [ExpectedException(typeof(NotEnoughFundsException))]
54     public void TestTransferWithTooHighValue()
55     {
56         double amountToTransferTooHigh = 10;
57         ba.Transfer(123, amountToTransferTooHigh);
58     }
59
60     [Test]
61     [ExpectedException(typeof(ArgumentException))]
62     public void TestTransferWithTooSmallValue()
63     {
64         double amountToTransferTooSmall = -10;
65         ba.Transfer(123, amountToTransferTooSmall);
66     }
67
68     [Test]
69     public void TestOrdinaryFunctions()
70     {
71         double currentValue = ba.Balance;
72         double deposit = 50.0;
73         currentValue += deposit;
74         ba.Deposit(deposit);
75         Assert.AreEqual(currentValue, ba.Balance);
76
77         double withdraw = 25.0;
78         currentValue -= withdraw;
79         ba.Withdraw(withdraw);
80         Assert.AreEqual(currentValue, ba.Balance);
81     }

```



```

82         double transfer = 10.50;
83         currentValue -= transfer;
84         ba.Transfer(123, transfer);
85         Assert.AreEqual(currentValue, ba.Balance);
86     }
87
88     [Test]
89     public void TestDepositWithdrawSameAmount()
90     {
91         double amount = 50;
92         double value = 0;
93         ba.Deposit(amount);
94         ba.Withdraw(amount);
95         Assert.AreEqual(value, ba.Balance);
96     }
97
98     [Test]
99     public void TestTransferAllMoney()
100    {
101        double amount = 50;
102        double value = 0;
103        ba.Deposit(amount);
104        ba.Transfer(0, amount);
105        Assert.AreEqual(value, ba.Balance);
106    }
107
108    [Test]
109    public void TestStringParsing()
110    {
111        BankAccount parseAccount = new BankAccount();
112
113        String parseOne = "123456,Rambo,500.50";
114        BankAccount ba1 = parseAccount.ParseFromString(parseOne);
115
116        String parseTwo = "000000,, ";
117        BankAccount ba2 = parseAccount.ParseFromString(parseTwo);
118    }
119
120    [Test]
121    [ExpectedException(typeof(FormatException))]
122    public void TestStringParsingInvalidValues()
123    {
124        BankAccount parseAccount = new BankAccount();
125
126        String parseOne = "abcd,Rambo,50.50";
127        BankAccount ba1 = parseAccount.ParseFromString(parseOne);
128    }
129 }
130 }

```

Listing 22: BankAccountTest.cs

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using ApplicationLogic;
5 using ApplicationLogic.Model;
6 using NUnit.Framework;
7 namespace NUnit_Tests
8 {
9     [TestFixture()]
10    public class FileHandlerTest
11    {
12        [Test]
13        [ExpectedException(typeof(NoFilePathSetException))]
14        public void TestWriteToFileWithoutPath ()
15        {
16            FileHandler<StockItem> fh = new FileHandler<StockItem>();
17            fh.SaveToFile(new List<StockItem>());
18        }
19
20        [Test]
21        [ExpectedException(typeof(NoFilePathSetException))]
22        public void TestReadFromFileWithoutPath()
23        {
24            FileHandler<StockItem> fh = new FileHandler<StockItem>();
25            fh.LoadFromFile(new StockItem());
26        }
27    }
28 }

```

Listing 23: FileHandlerTest.cs

```

1 i>using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using ApplicationLogic.Model;
6 using NUnit.Framework;
7
8 namespace NUnit_Tests
9 {
10    [TestFixture]
11    public class StockItemTest
12    {
13        private StockItem si;
14
15        [SetUp]
16        public void SetUp()
17        {
18            si = new StockItem("1234", "Test", "Test", 10.0, 5, 5);
19        }

```

```

20
21 [Test]
22 [ExpectedException(typeof(ArgumentException))]
23 public void TestLessThan0Cost()
24 {
25     si = new StockItem("1234", "Test", "Test", -1.0, 5, 5);
26 }
27
28 [Test]
29 [ExpectedException(typeof(ArgumentException))]
30 public void TestLessThan0CurrentStock()
31 {
32     si = new StockItem("1234", "Test", "Test", 1.0, -1, 5);
33 }
34
35 [Test]
36 [ExpectedException(typeof(ArgumentException))]
37 public void TestLessThan0RequiredStock()
38 {
39     si = new StockItem("1234", "Test", "Test", 1.0, 5, -1);
40 }
41
42 [Test]
43 public void TestIsValidStockCode()
44 {
45     bool validSC = StockItem.IsValidStockCode("1234");
46     Assert.IsTrue(validSC);
47     bool tooLongSC = StockItem.IsValidStockCode("123456");
48     Assert.IsFalse(tooLongSC);
49     bool stringSC = StockItem.IsValidStockCode("test");
50     Assert.IsFalse(stringSC);
51 }
52
53 [Test]
54 [ExpectedException(typeof(ArgumentNullException))]
55 public void TestIsValidStockCodeRaiseException()
56 {
57     StockItem.IsValidStockCode(null);
58 }
59
60 [Test]
61 [ExpectedException(typeof(ArgumentException))]
62 public void TestCreateStockItemInvalidStockCode()
63 {
64     String invalidStockCode = "ooooi";
65     StockItem si = new StockItem(invalidStockCode, "", "", 0.0, 0, 0);
66 }
67
68 [Test]
69 [ExpectedException(typeof(ArgumentException))]
70 public void TestCreateStockItemInvalidCost()

```

```

71     {
72         double invalidCost = -1.0;
73         StockItem si = new StockItem("0001", "", "", invalidCost, 0, 0);
74     }
75
76     [Test]
77     [ExpectedException(typeof(ArgumentException))]
78     public void TestCreateStockItemInvalidRequiredStock()
79     {
80         int invalidStock = -1;
81         StockItem si = new StockItem("0001", "", "", 0.0, invalidStock, 0);
82     }
83
84     [Test]
85     [ExpectedException(typeof(ArgumentException))]
86     public void TestCreateStockItemInvalidCurrentStock()
87     {
88         int invalidStock = -1;
89         StockItem si = new StockItem("0001", "", "", 0.0, 0, invalidStock);
90     }
91
92     [Test]
93     public void TestStringParsing()
94     {
95         StockItem parseItem = new StockItem();
96
97         String parseOne = "0001,Pencil Holder,John Rambo,5.50,10,15";
98
99         StockItem si = parseItem.ParseFromString(parseOne);
100        Assert.IsNotNull(si);
101
102        String parseTwo = "0001,,John Rambo,5.50,10,15";
103
104        StockItem si2 = parseItem.ParseFromString(parseTwo);
105        Assert.IsNotNull(si2);
106
107        String parseThree = "0001,,,,,";
108
109        StockItem si3 = parseItem.ParseFromString(parseThree);
110        Assert.IsNotNull(si3);
111    }
112
113    [Test]
114    [ExpectedException(typeof(FormatException))]
115    public void TestStringParsingInvalidValues()
116    {
117        StockItem parseItem = new StockItem();
118
119        String parseOne = "0001,Pencil Holder,John Rambo,abc,10,15";
120        StockItem si = parseItem.ParseFromString(parseOne);
121    }

```

```

122     }
123 }

```

Listing 24: StockItemTest.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using NUnit.Framework;
6  using Moq;
7  using ApplicationLogic.Model;
8
9  namespace NUnit_Tests.ApplicationLogic
10 {
11     [TestFixture]
12     public class AppDataManagerTest
13     {
14         private AppDataManager _Manager;
15         private StockItem _Stock;
16         private BankAccount _Account;
17
18         [SetUp]
19         public void SetUp()
20         {
21             this._Manager = new AppDataManager();
22             this._Stock = new StockItem();
23             this._Account = new BankAccount();
24         }
25
26         [Test]
27         public void TestAddStockItem()
28         {
29             this._Manager.CreateNewStockItem();
30             Assert.AreEqual(1, this._Manager.StockItems.Count);
31         }
32
33         [Test]
34         public void TestAddBankAccount()
35         {
36             this._Manager.CreateNewBankAccount();
37             Assert.AreEqual(1, this._Manager.BankAccounts.Count);
38         }
39
40         [Test]
41         public void TestRemoveStockItem()
42         {
43             this._Manager.CreateNewStockItem();
44             Assert.AreEqual(1, this._Manager.StockItems.Count);
45             StockItem remove = this._Manager.StockItems.ElementAt(0);
46             this._Manager.DeleteStockItem(remove);

```

```

47         Assert.AreEqual(0, this._Manager.StockItems.Count);
48     }
49
50     [Test]
51     public void TestRemoveBankAccount()
52     {
53         this._Manager.CreateNewBankAccount();
54         Assert.AreEqual(1, this._Manager.BankAccounts.Count);
55         BankAccount remove = this._Manager.BankAccounts.ElementAt(0);
56         this._Manager.DeleteBankAccount(remove);
57         Assert.AreEqual(0, this._Manager.BankAccounts.Count);
58     }
59
60     [Test]
61     [ExpectedException(typeof(ArgumentException))]
62     public void TestRemoveStockItemNotPresent()
63     {
64         StockItem si = new StockItem();
65         this._Manager.DeleteStockItem(si);
66     }
67
68     [Test]
69     [ExpectedException(typeof(ArgumentException))]
70     public void TestRemoveBankAccountNotPresent()
71     {
72         BankAccount ba = new BankAccount();
73         this._Manager.DeleteBankAccount(ba);
74     }
75
76     [Test]
77     public void TestCorrectSequenceOfOrdering()
78     {
79         var mockBa = new Mock<BankAccount>();
80         var mockSi = new Mock<StockItem>();
81
82         mockBa.Setup(ba => ba.Balance).Returns(50.0);
83
84         this._Manager.BankAccounts.Add(mockBa.Object);
85         this._Manager.StockItems.Add(mockSi.Object);
86         this._Manager.OrderItem(mockSi.Object, mockBa.Object, 0);
87
88         mockBa.VerifyGet(ba => ba.Balance);
89         mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.AtMostOnce());
90     }
91
92     [Test]
93     [ExpectedException(typeof(NotEnoughFundsException))]
94     public void TestCorrectSequenceOnInvalidFunds()
95     {
96         var mockBa = new Mock<BankAccount>();
97         var mockSi = new Mock<StockItem>();

```

```
98
99     mockBa.Setup(ba => ba.Balance).Returns(0.0);
100    mockSi.Setup(si => si.UnitCost).Returns(10.0);
101
102    this._Manager.BankAccounts.Add(mockBa.Object);
103    this._Manager.StockItems.Add(mockSi.Object);
104    this._Manager.OrderItem(mockSi.Object, mockBa.Object, 1);
105
106    mockBa.VerifyGet(ba => ba.Balance);
107    mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.Never());
108    }
109 }
110 }
```

Listing 25: AppDataManagerTest.cs

BIBLIOGRAPHY

- Balzert, Helmut (2009). *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Heidelberg: Spektrum. ISBN: 9783827417053.
- Boodhoo, Jean-Paul (2006). *Design Patterns: Model View Presenter*. English. Microsoft. URL: <http://msdn.microsoft.com/en-us/magazine/cc188690.aspx> (visited on 19/10/2010).
- Dorman, Scott (2010). *Sams Teach Yourself Visual CSharp®2010 in 24 Hours: Complete Starter Kit*. Sams Publishing. ISBN: 978-0-672-33101-5.
- Freeman, Eric and Elisabeth Freeman (2004). *Head First - Design Pattern*. Ed. by Mike Loukides. O'Reilly. ISBN: 0-596-00712-4.
- Noyes, Brian (2006). *Data Binding With Windows Forms 2.0 - Programming Smart Client Data Applications With .NET*. Addison Wesley Professional. ISBN: 978-0-321-26892-1.
- Sommerville, Ian (2006). *Software Engineering*. 8th ed. Addison Wesley. ISBN: 9780321210265.
- Stellman, Andrew and Jennifer Greene (2010). *Head First CSharp*. O'Reilly. ISBN: 978-1-449-38034-2.