

SYSTEMS PROGRAMMING AND SCRIPTING

FLORIAN BERGMANN

Assessment One: Stock Manager

[October 19, 2010 at 14:19]

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
4 USER GUIDE	6
4.1 Manage stock items and bank accounts	6
4.2 Placing an order	7
4.3 Importing & exporting data	8
4.3.1 File menu	8
4.3.2 Menu-bar icon	8
5 DEVELOPER GUIDE	10
5.1 Architectural overview	10
5.2 UserInterface	11
5.3 ApplicationLogic	13
5.3.1 FileHandler	14
5.3.2 Error handling	15
6 TESTING	17
7 CONCLUSIONS	18
 II APPENDIX	 19
A APPENDIX: SOURCE CODE	20
A.1 View	20
A.2 Application Logic	32
A.2.1 Interfaces Package	32
A.2.2 Model Package	35
A.2.3 Presenter Package	55
A.3 Tests	61
 BIBLIOGRAPHY	 67

LIST OF FIGURES

Figure 1	Main Window	6
Figure 2	Add a stock item.	7
Figure 3	Add a bank account.	7
Figure 4	Depositing and withdrawing money	7
Figure 5	Selection of items.	7
Figure 6	Placing an order without the needed funds	8
Figure 7	Saving via file menu	8
Figure 8	Settings window	9
Figure 9	Architecture overview with model-view-presenter (MVP) . . .	10
Figure 10	Abstract overview of project application logic	14
Figure 11	Sequence diagram of input validation	16

LIST OF TABLES

LISTINGS

Listing 1	Data Binding of view and model	11
Listing 2	Example interface IStockItemView	12
Listing 3	Interface ICongregateView	12
Listing 4	Interface ICSVSerializable	14
Listing 5	Validate method of StockItem	15
Listing 6	MainWindow.cs	20
Listing 7	Settings.cs	31
Listing 8	IBankAccountView.cs	32
Listing 9	ICongregateView.cs	33
Listing 10	ICSVSerializable.cs	33

Listing 11	IStockItemView.cs	34
Listing 12	IViewModel.cs	34
Listing 13	AppDataManager.cs	35
Listing 14	BankAccount.cs	42
Listing 15	ErrorMessage.cs	46
Listing 16	ErrorMessageCollection.cs	47
Listing 17	FileHandler.cs	48
Listing 18	NoFilePathSetException.cs	49
Listing 19	NotEnoughFundsException.cs	50
Listing 20	StockItem.cs	50
Listing 21	CongregatePresenter.cs	55
Listing 22	BankAccountTest.cs	61
Listing 23	FileHandlerTest.cs	63
Listing 24	StockItemTest.cs	64

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

Part I

DEVELOPMENT OF A STOCK MANAGER APPLICATION

INTRODUCTION

In 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. Naturally part is essential for end users.

The chapters [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

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 following 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 level*.

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 files 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 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.

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 allow adding and deleting.
- MSIO3: Implemented as StockItem class.
- MSIO4: Implemented in manager-class.
- MBA01: Implemented in BankAccount class with getters and setters.
- MBA02: Fake-method for ordering: will adjust account balance, but not transfer money.
- MBA03: Implemented in manager-class: takes care of Atomicity of request.
- MDA01: Implemented in FileHandler-class and StockItem-class.
- MDA02: Implemented in FileHandler-class.
- MDA03: Implemented in StockItem-class.
- MDA04: Implemented in StockItem-class.
- MDA04: Implemented in StockItem-class.
- MDA05: Verified via testing.
- GUI01: Implemented via WinForms.
- GUI02: 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.
- BANKACCOUNT 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

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 three use cases.

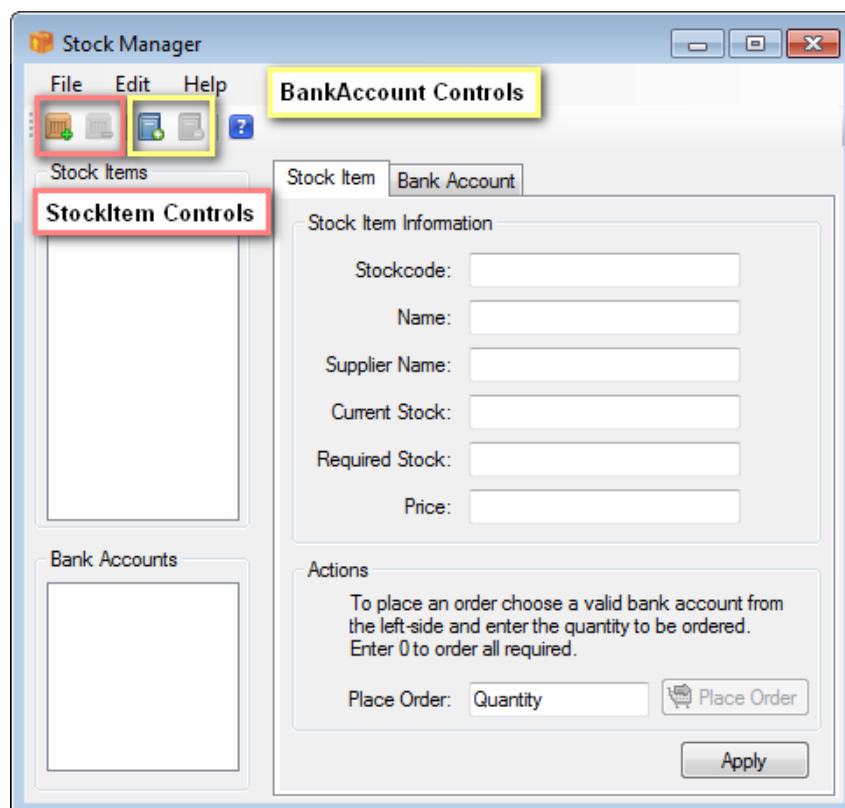


Figure 1: Main Window

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



Figure 2: Add a stock item.

Icon to add a new stock item to the application: the item will be inserted into the stock item list with dummy values.

Icon to add a new bank account to the application: the account will be inserted into the bank account list with dummy values.



Figure 3: 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.

Figure 4: Depositing and withdrawing money

4.2 PLACING AN ORDER

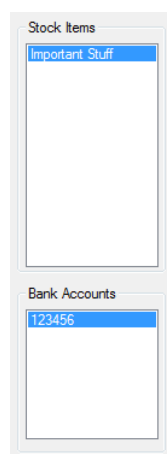


Figure 5: Selection of items.

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 can be entered inside the *quantity*-box: either the amount of items to be ordered, or 0. 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.

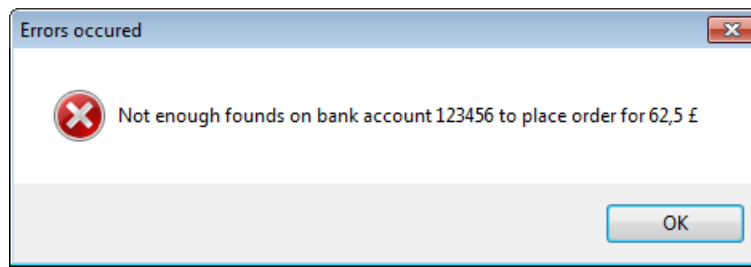


Figure 6: Placing an order without the needed funds

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.

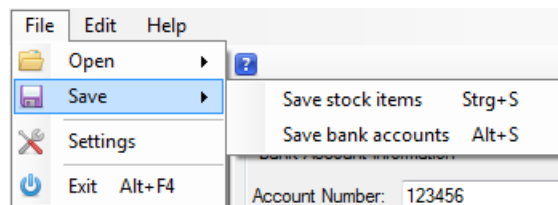


Figure 7: Saving via file menu

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.

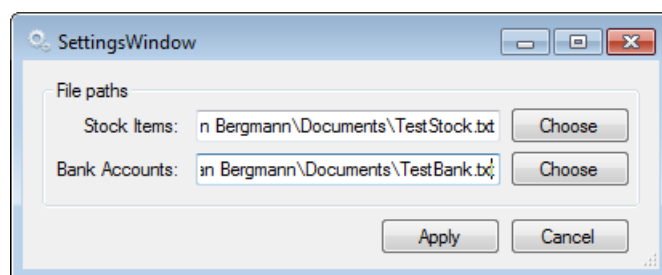


Figure 8: Settings window

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

To allow further development of the application, the design will be described in a top down approach, starting from a high level overview and going into details only where necessary.

5.1 ARCHITECTURAL OVERVIEW

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

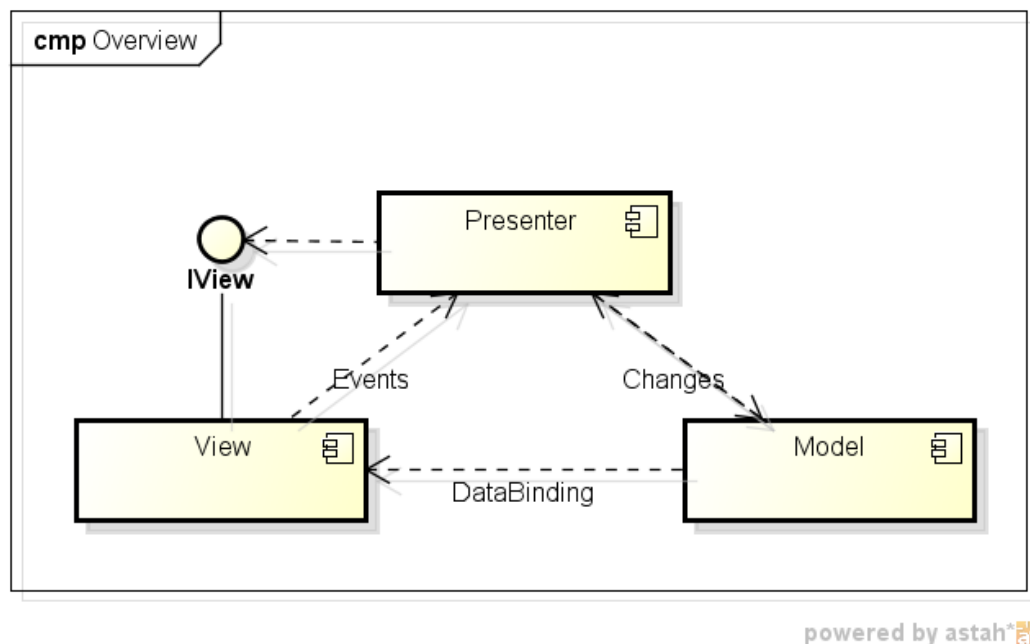


Figure 9: Architecture overview with MVP

This separation allows the view to be decoupled from the presenter and the model, as the presenter only communicates with the view over a well-defined interface (IView), whereas the view communicates only with provided methods from the presenter.

When changes occur in the model, the view will be notified via the data-binding mechanism of 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.

These two packages will be described in-depth:

5.2 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, "
            SupplierName", false, DataSourceUpdateMode.Never);
18         currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
19             ", false, DataSourceUpdateMode.Never);
20         reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
21             ", false, DataSourceUpdateMode.Never);
22         priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
23             , DataSourceUpdateMode.Never);

```

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.

```

18         bankAccountsListBox.DataSource = _Model.BankAccounts;
19         bankAccountsListBox.DisplayMember = "AccountNumber";
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 that will allow us to handle erroneous input.

Another important architectural aspect of the view is that the necessary interfaces for the presenter needs to be implemented: 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, as well as provide the necessary application logic to order items, deposit and withdraw money as well as method to display possible validation errors.

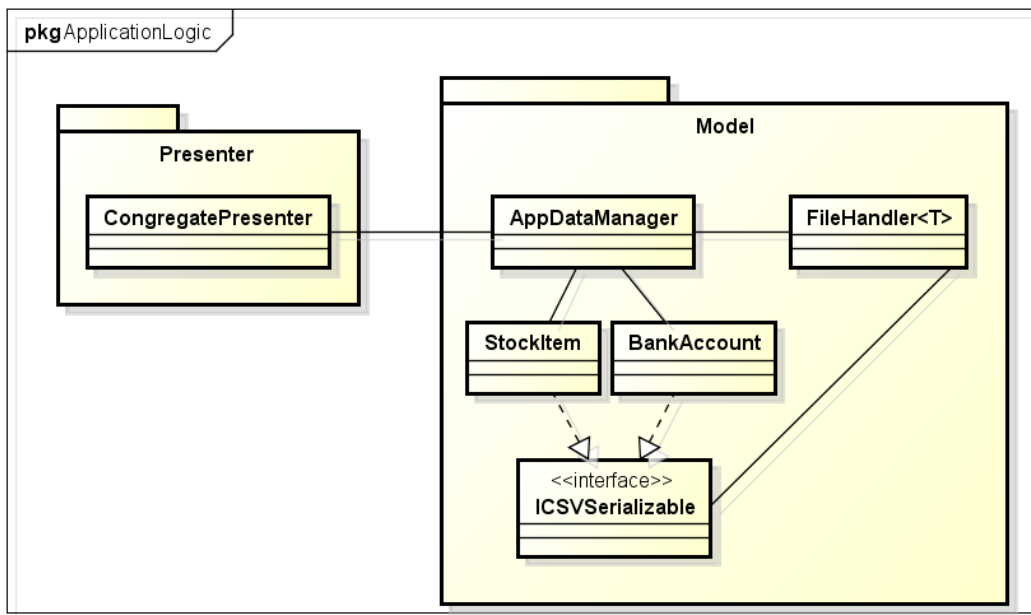
Should new views be added an interface should be provided that guarantees the separation between the view and the presenter.

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).

5.3 APPLICATIONLOGIC

The application logic project hosts the presenter(s) as well as the models.

An overview of the relation between the classes shall be provided for orientation purposes:



powered by astah®

Figure 10: Abstract overview of project application logic

It can be seen that the AppDataManager-class works as a *facade* for the rest of the model.

Noteworthy implementation in this package are the FileHandler and the realisation of error handling.

5.3.1 FileHandler

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  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 4: Interface ICSVSerializable

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

5.3.2 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 display the occurred errors.

As an example the code of the StockItem'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             ErrorMessageCollection.Add(new ErrorMessage("Need an item name."));
10        }
11        if (String.IsNullOrEmpty("supplierName"))
12        {
13            ErrorMessageCollection.Add(new ErrorMessage("Need a supplier name."));
14        }
15        if (unitCost < 0.0)
16        {
17            ErrorMessageCollection.Add(new ErrorMessage("Unit costs must be greater or
                equal 0."));
18        }
19        if (required < 0)
20        {
21            ErrorMessageCollection.Add(new ErrorMessage("Required must be greater or equal
                0."));
22        }

```

```

23     if (currentStock < 0)
24     {
25         ErrorMessage.Add(new ErrorMessage("Current must be greater or equal
26         o."));
27     }
28     return ErrorMessage.Count == 0;
  
```

Listing 5: Validate method of StockItem

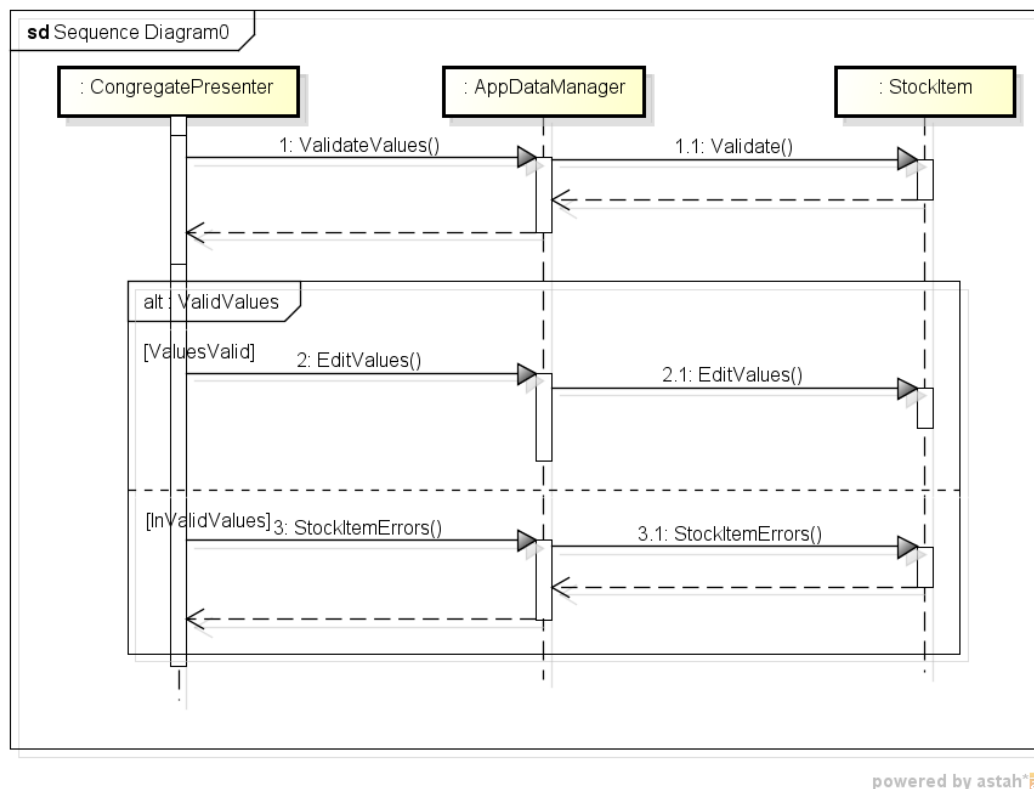


Figure 11: Sequence diagram of input validation

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 three main-classes:

1. StockItem
2. BankAccount
3. FileHandler

CONCLUSIONS

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,
19         IBankAccountView
20     {
21         private CongregatePresenter _Presenter;
22         private IViewModel _Model;
23         private List<Control> backgroundColorChanged;
24
25         private const int STOCKITEMTAB = 0;
26         private const int BANKACCOUNTTAB = 1;
27
28         /// <summary>
29         /// Initializes a new instance of the <see cref="MainWindow"/> class.
30         /// </summary>
31         public MainWindow()
32         {
33             InitializeComponent();
34             this._Model = new AppDataManager();
35             _Presenter = new CongregatePresenter(this, this, this, this._Model);
36             this.backgroundColorChanged = new List<Control>();
37             LoadFilePathSettings();
38             SetUpDataBindings();
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 }
82
83 private void addBankAccountToolStripMenuItem_Click(object sender, EventArgs e)
84 {
85     this._Presenter.CreateNewBankAccount();
86 }
87
88 private void deleteBankAccountToolStripMenuItem_Click(object sender, EventArgs
89     e)
90 {
91     this._Presenter.DeleteBankAccount();
92 }

```

```

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

```

```

137         control.BackColor = Color.White;
138     }
139 }
140
141 private void applyBankAccountButton_Click(object sender, EventArgs e)
142 {
143     this.ResetColoring();
144     this._Presenter.EditBankAccount();
145 }
146
147 void PlaceOrderButton_Click(object sender, EventArgs e)
148 {
149     this._Presenter.OrderItem();
150 }
151
152 private void depositButton_Click(object sender, EventArgs e)
153 {
154     this._Presenter.Deposit();
155 }
156
157 private void withdrawButton_Click(object sender, EventArgs e)
158 {
159     this._Presenter.Withdraw();
160 }
161
162 private void exitToolStripMenuItem_Click(object sender, EventArgs e)
163 {
164     this._Presenter.CloseApplication();
165 }
166
167 private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
168 {
169     SettingsWindow sw = new SettingsWindow();
170     sw.ShowDialog();
171     this.LoadFilePathSettings();
172 }
173
174 private void saveStripButton_Click(object sender, EventArgs e)
175 {
176     this._Presenter.SaveBankAccountsToFile();
177     this._Presenter.SaveStockItemsToFile();
178 }
179
180 #endregion
181
182 #region IStockItemView
183
184 /// <summary>
185 /// Gets the current stock.
186 /// </summary>
187 /// <value>The current stock.</value>

```

```

188 public int CurrentStock
189 {
190     get
191     {
192         int currentStock;
193         try
194         {
195             currentStock = int.Parse(currStockTextBox.Text);
196             return currentStock;
197         }
198         catch (FormatException e)
199         {
200             this.DisplayError(currStockTextBox);
201             throw;
202         }
203     }
204 }
205
206 /// <summary>
207 /// Gets the required stock.
208 /// </summary>
209 /// <value>The required stock.</value>
210 public int RequiredStock
211 {
212     get
213     {
214         int requiredStock;
215         try
216         {
217             requiredStock = int.Parse(reqStockTextBox.Text);
218             return requiredStock;
219         }
220         catch (FormatException e)
221         {
222             this.DisplayError(reqStockTextBox);
223             throw;
224         }
225     }
226 }
227
228 /// <summary>
229 /// Gets the stock code.
230 /// </summary>
231 /// <value>The stock code.</value>
232 public string StockCode
233 {
234     get { return stockCodeTextBox.Text; }
235 }
236
237 /// <summary>
238 /// Gets the name of the supplier.

```

```

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

```

```

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

```

```

341     /// <value>The stock item.</value>
342     public StockItem StockItem
343     {
344         get { return (StockItem)this.stockItemsListBox.SelectedItem; }
345     }
346
347     /// <summary>
348     /// Gets the bank account.
349     /// </summary>
350     /// <value>The bank account.</value>
351     public BankAccount BankAccount
352     {
353         get { return (BankAccount)this.bankAccountsListBox.SelectedItem; }
354     }
355
356     /// <summary>
357     /// Gets the quantity value.
358     /// </summary>
359     /// <value>The quantity.</value>
360     public int Quantity
361     {
362         get
363         {
364             int quan = 0;
365             try
366             {
367                 quan = int.Parse(quantityTextBox.Text);
368                 return quan;
369             }
370             catch (FormatException e)
371             {
372                 this.DisplayError(quantityTextBox);
373                 throw;
374             }
375         }
376     }
377
378     /// <summary>
379     /// Gets the deposit value.
380     /// </summary>
381     /// <value>The deposit.</value>
382     public double Deposit
383     {
384         get
385         {
386             double deposit;
387             try
388             {
389                 deposit = double.Parse(depositQuantityTextBox.Text);
390                 return deposit;
391             }

```

```

392         catch (FormatException e)
393         {
394             this.DisplayError(depositQuantityTextBox);
395             throw;
396         }
397     }
398 }
399
400 /// <summary>
401 /// Gets the withdraw value.
402 /// </summary>
403 /// <value>The withdraw.</value>
404 public double Withdraw
405 {
406     get
407     {
408         double withdraw;
409         try
410         {
411             withdraw = double.Parse(withdrawQuantityTextBox.Text);
412             return withdraw;
413         }
414         catch (FormatException e)
415         {
416             this.DisplayError(withdrawQuantityTextBox);
417             throw;
418         }
419     }
420 }
421
422 /// <summary>
423 /// Displays the validation errors.
424 /// </summary>
425 /// <param name="errorCollection">The error collection.</param>
426 public void DisplayValidationErrors(ErrorMessageCollection errorCollection)
427 {
428     MessageBox.Show(errorCollection.ToString(), "Errors occurred",
429         MessageBoxButtons.OK, MessageBoxIcon.Error);
430 }
431
432 /// <summary>
433 /// Asks fir confirmation of a deletion.
434 /// </summary>
435 /// <returns></returns>
436 public bool ConfirmDelete()
437 {
438     DialogResult result = MessageBox.Show("Are you sure you want to delete
439         this item?", "Confirm delete", MessageBoxButtons.YesNo, MessageBoxIcon
440         .Question);
441     return result == DialogResult.Yes;
442 }

```



```

440
441     /// <summary>
442     /// Asks fir confirmation of closing the application.
443     /// </summary>
444     /// <returns></returns>
445     public bool ConfirmClose()
446     {
447         DialogResult result = MessageBox.Show("Are you sure you want to close the
            application?", "Confirm close", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);
448         return result == DialogResult.Yes;
449     }
450
451     #endregion
452
453     #region Private methods
454
455     /// <summary>
456     /// Switches the BankAccount Controls depending on the selection.
457     /// </summary>
458     /// <param name="enabled">True if controls shall be enabled, false otherwise
        .</param>
459     private void SwitchBankAccountControls(bool enabled)
460     {
461         this.deleteBankAccountToolStripMenuItem.Enabled = enabled;
462         this.deleteBankAccountToolStripButton.Enabled = enabled;
463     }
464
465     /// <summary>
466     /// Switches the StockItem Controls depending on the selection.
467     /// </summary>
468     /// <param name="enabled">True if controls shall be enabled, false otherwise
        .</param>
469     private void SwitchStockItemControls(bool enabled)
470     {
471         this.deleteStockItemToolStripButton.Enabled = enabled;
472         this.deleteStockItemToolStripMenuItem.Enabled = enabled;
473     }
474
475     private void DisplayError(Control form)
476     {
477         backgroundColorChanged.Add(form);
478         form.BackColor = Color.MistyRose;
479     }
480
481     private void LoadFilePathSettings()
482     {
483         Settings settings = Settings.Default;
484
485         String filePathStockItems = settings.StockItemFilePath;
486         String filePathBankAccounts = settings.BankAccountFilePath;

```

```

487         if (!String.IsNullOrEmpty(filePathStockItems))
488         {
489             this._Presenter.SetupStockItemFilePath(filePathStockItems);
490         }
491         if (!String.IsNullOrEmpty(filePathBankAccounts))
492         {
493             this._Presenter.SetupBankAccountsFilePath(filePathBankAccounts);
494         }
495     }
496
497     private void SetupDataBindings()
498     {
499         stockItemsListBox.DataSource = _Model.StockItems;
500         stockItemsListBox.DisplayMember = "Name";
501
502         /*
503          * The datasourceupdate mode is set to "Never".
504          * This leads to the ability to enforce the use of the presenter to update
505          * the values in the model.
506          * This way the validation errors can be handled by the presenter thus
507          * leading to better separation of concerns.
508          */
509         stockCodeTextBox.DataBindings.Add("Text", _Model.StockItems, "StockCode",
510             false, DataSourceUpdateMode.Never);
511         itemNameTextBox.DataBindings.Add("Text", _Model.StockItems, "Name", false,
512             DataSourceUpdateMode.Never);
513         supplierNameTextBox.DataBindings.Add("Text", _Model.StockItems, "
514             SupplierName", false, DataSourceUpdateMode.Never);
515         currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
516             ", false, DataSourceUpdateMode.Never);
517         reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
518             ", false, DataSourceUpdateMode.Never);
519         priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
520             , DataSourceUpdateMode.Never);
521
522         bankAccountsListBox.DataSource = _Model.BankAccounts;
523         bankAccountsListBox.DisplayMember = "AccountNumber";
524
525         accountNumberTextBox.DataBindings.Add("Text", _Model.BankAccounts, "
526             AccountNumber", false, DataSourceUpdateMode.Never);
527         nameTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Surname", false
528             , DataSourceUpdateMode.Never);
529         balanceTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Balance",
530             false, DataSourceUpdateMode.Never);
531     }
532
533     private void quantityTextBox_TextChanged(object sender, EventArgs e)
534     {
535         String newText = quantityTextBox.Text;
536         int parseInt = 0;
537         if (int.TryParse(newText, out parseInt))

```

```

527         {
528             placeOrderButton.Enabled = true;
529         }
530         else
531         {
532             placeOrderButton.Enabled = false;
533         }
534     }
535
536     #endregion
537 }
538 }

```

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 data and persistent.
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,
121                             string name, int currentStock, int reqStock, double price)
122 {
123     if (si != null)
124     {
125         si.EditStockItem(stockCode, name, supplier, price, reqStock,
126                         currentStock);
127     }
128     else
129     {
130         throw new ArgumentNullException("Stock item to edit not present.");
131     }
132 }
133
134 /// <summary>
135 /// Attempts to edit specified BankAccount in the BankAccount collection.
136 /// Throws an ArgumentException if the account can not be found.
137 /// </summary>
138 /// <param name="ba">BankAccount to be edited</param>
139 /// <param name="surname">New surname</param>
140 /// <param name="accountNumber">New accountnumber</param>
141 internal void EditBankAccount(BankAccount ba, string surname, int
142                               accountNumber)
143 {
144     if (ba != null)
145     {
146         ba.EditBankAccount(surname, accountNumber);
147     }
148     else
149     {
150         throw new ArgumentNullException("Bank account to edit not present.");
151     }
152 }

```

```

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 vald. 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                 buyExcessStock = true;
234             }
235             double priceOfOrder = quantity * si.UnitCost;
236             if (priceOfOrder < ba.Balance)
237             {
238                 ba.Transfer(1, priceOfOrder);
239                 si.CurrentStock += quantity;
240                 /*
241                  * Allow the user to buy more than needed.
242                  */
243                 if (buyExcessStock)
244                 {
245                     if (quantity > si.RequiredStock)
246                     {
247                         si.RequiredStock = 0;
248                     }
249                 }
250             }
251         }
252     }

```

```

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

```

```

290     }
291
292     /// <summary>
293     /// Will reload the StockItem collection from the specified file.
294     /// Will overwrite the currently existing StockItem collection.
295     /// </summary>
296     /// <param name="filePath"></param>
297     internal void LoadStockItemsFromFile(string filePath)
298     {
299         this.StockItemHandler.ReadFilePath = filePath;
300         IList<StockItem> stockItems = StockItemHandler.LoadFromFile(new StockItem
301             ());
302         this.StockItems.Clear();
303         foreach (StockItem item in stockItems)
304         {
305             this.StockItems.Add(item);
306         }
307     }
308
309     /// <summary>
310     /// Will reload the BankAccount collection from the specified file.
311     /// Will overwrite the currently existing BankAccount collection.
312     /// </summary>
313     /// <param name="filePath">The path to the file</param>
314     internal void LoadBankAccountsFromFile(string filePath)
315     {
316         this.BankAccountHandler.ReadFilePath = filePath;
317         IList<BankAccount> bankAccounts = BankAccountHandler.LoadFromFile(new
318             BankAccount());
319         this.BankAccounts.Clear();
320         foreach (BankAccount item in bankAccounts)
321         {
322             this.BankAccounts.Add(item);
323         }
324     }
325
326     /// <summary>
327     /// Will save the current StockItem collection to the specified file.
328     /// </summary>
329     /// <param name="filePath">The path to the file</param>
330     internal void SaveStockItemsToFile(string filePath)
331     {
332         this.StockItemHandler.WriteFilePath = filePath;
333         this.StockItemHandler.SaveToFile(this.StockItems);
334     }
335
336     /// <summary>
337     /// Will save the current BankAccount collection to the specified file.
338     /// </summary>
339     /// <param name="filePath">The path to the file</param>
340     internal void SaveBankAccountsToFile(string filePath)

```

```

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

```

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>
20         /// <value>The account number.</value>
21         public int AccountNumber {
22             get { return _AccountNumber; }
23         }
24     }
25 }

```

```

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 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 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 ArgumentException ("Not enough
94                 funds on bank account to withdraw.");
95             }
96         } else {
97             throw new ArgumentException ("There are not enough
98             funds present to fulfill the required action.");
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.
119     /// NOTE: This method is a fake to simulate "real" banking. The money
120     /// will not be transfered to any account.
121     /// </summary>

```



```

116     /// <param name="amount">Amount to be transfered - must be greater
117     than 0.</param>
118     /// <param name="accountNumber">Account number to transfer the money
119     to.</param>
120     public void Transfer (int accountNumber, double amount)
121     {
122         if (amount >= 0.0) {
123             if (this.Balance > amount) {
124                 this.Balance -= amount;
125                 // TODO: In reality: fancy logic to transfer
126                 money.
127             } else {
128                 throw new ArgumentException ("There are not
129                 enough funds present to fulfill the
130                 required action.");
131             }
132         } else {
133             throw new ArgumentException ("It is not possible to
134             transfer funds from another account to yours.");
135         }
136     }
137
138     /// <summary>
139     /// Validates a set of possible changes to a BankAccount.
140     /// </summary>
141     /// <param name="accountNumber">AccountNumber to be verified</param>
142     /// <param name="surname">Surname to be verified</param>
143     /// <returns>True if the values would be valid, false otherwise.</
144     returns>
145     static internal bool Validate (int accountNumber, String surname)
146     {
147         if (String.IsNullOrEmpty (surname)) {
148             ErrorMessage.Add (new ErrorMessage ("Need the name of
149             the account owner."));
150         }
151         if (accountNumber <= 0) {
152             ErrorMessage.Add (new ErrorMessage ("Need a valid
153             account number: greater o."));
154         }
155         return ErrorMessage.Count == 0;
156     }
157
158     public event PropertyChangedEventHandler PropertyChanged;
159
160     private void NotifyPropertyChanged (String info)
161     {
162         if (PropertyChanged != null) {
163             PropertyChanged (this, new PropertyChangedEventArgs (
164             info));
165         }
166     }

```

```

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         String accountNumber = split[0];
185         String surname = split[1];
186         String balance = split[2];
187         int accNumber = 0;
188         double bal = 0;
189         if (!String.IsNullOrEmpty (accountNumber)) {
190             accNumber = int.Parse (accountNumber);
191         }
192         if (!String.IsNullOrEmpty (balance)) {
193             bal = double.Parse (balance);
194         }
195         return new BankAccount (accNumber, surname, bal);
196     }
197 }

```

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>
34         /// <param name="elements">Collection to be saved to file.</param>
35         public void SaveToFile(IList<T> elements)
36         {
37             if (!String.IsNullOrEmpty(this.WriteFilePath))
38             {
39                 FileStream writeFile = File.Open(WriteFilePath, FileMode.Create);
40                 using (StreamWriter sw = new StreamWriter(writeFile))
41                 {
42                     foreach (T item in elements)
43                     {
44                         sw.Write(item.CsvRepresentation());
45                         sw.Write("\n");
46                     }
47                 }
48             }
49         }
50     }
51 }

```

```

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     else
85     {
86         throw new NoFilePathSetException("No file path to write to set.");
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 String StockCode {

```

```

19         get { return _StockCode; }
20         set {
21             if (!IsValidStockCode (value)) {
22                 throw new ArgumentException ("Provided
                stockcode did not match designated format.
                ");
23             } else {
24                 this._StockCode = value;
25                 this.NotifyPropertyChanged ("StockCode");
26             }
27         }
28     }
29
30     private String _Name;
31     public String Name {
32         get { return _Name; }
33         set {
34             _Name = value;
35             this.NotifyPropertyChanged ("Name");
36         }
37     }
38
39     private String _SupplierName;
40     public String SupplierName {
41         get { return _SupplierName; }
42         set {
43             _SupplierName = value;
44             this.NotifyPropertyChanged ("SupplierName");
45         }
46     }
47
48     private double _UnitCost;
49     public double UnitCost {
50         get { return _UnitCost; }
51         private set {
52             if (value < 0.0) {
53                 throw new ArgumentException ("Price can not be
                lower than 0.");
54             } else {
55                 _UnitCost = value;
56                 this.NotifyPropertyChanged ("UnitCost");
57             }
58         }
59     }
60 }
61
62 private int _RequiredStock;
63 public 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 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;
101     this.Name = name;
102     this.SupplierName = supplierName;
103     this.UnitCost = unitCost;
104     this.RequiredStock = required;
105     this.CurrentStock = currentStock;
106 }
107
108 /// <summary>
109 /// Checks if the stock code conforms to a certain format.
110 /// Current format is exactly four numbers, with leading 0 allowed.
111 /// </summary>
112 /// <param name="value">The string that must be checked against the
113     schema.</param>
114 /// <returns>True if the string conforms to the schema, false
115     otherwise.</returns>

```



```

112 public static bool IsValidStockCode (string value)
113 {
114     if (String.IsNullOrEmpty (value)) {
115         throw new ArgumentNullException ("Provided stockcode
116         was null or empty.");
117     } else {
118         // TODO: Do not use magic numbers in code
119         Regex regexp = new Regex (REGEX);
120         return regexp.IsMatch (value);
121     }
122 }
123
124 public void EditStockItem (String stockCode, String name, String
125     supplierName, double unitCost, int required, int currentStock)
126 {
127     this.StockCode = stockCode;
128     this.Name = name;
129     this.SupplierName = supplierName;
130     this.UnitCost = unitCost;
131     this.RequiredStock = required;
132     this.CurrentStock = currentStock;
133 }
134
135 /// <summary>
136 /// Validates a set of possible changes to a StockItem.
137 /// </summary>
138 /// <param name="stockCode">StockCodew to be verified</param>
139 /// <param name="name">Name to be verified</param>
140 /// <param name="supplierName">SupplierName to be verified</param>
141 /// <param name="unitCost">UnitCost to be verified</param>
142 /// <param name="required">RequiredStock to be verified</param>
143 /// <param name="currentStock">CurrentStock to be verified</param>
144 /// <returns>True if the values would be valid, false otherwise.</
145     returns>
146 public static bool Validate (String stockCode, String name, String
147     supplierName, double unitCost, int required, int currentStock)
148 {
149     if (String.IsNullOrEmpty (stockCode) || !IsValidStockCode (
150     stockCode)) {
151         ErrorMessage.Add (new ErrorMessage ("Need a stockcode
152         that adheres to the stockcode format: 4 numbers."
153         ));
154     }
155     if (String.IsNullOrEmpty (name)) {
156         ErrorMessage.Add (new ErrorMessage ("Need an item
157         name."));
158     }
159     if (String.IsNullOrEmpty (supplierName)) {
160         ErrorMessage.Add (new ErrorMessage ("Need a supplier
161         name."));

```

```

154         }
155         if (unitCost < 0.0) {
156             ErrorMessage.Add (new ErrorMessage ("Unit costs must
157                                     be greater or equal 0."));
158         }
159         if (required < 0) {
160             ErrorMessage.Add (new ErrorMessage ("Required must be
161                                     greater or equal 0."));
162         }
163         if (currentStock < 0) {
164             ErrorMessage.Add (new ErrorMessage ("Current must be
165                                     greater or equal 0."));
166         }
167         return ErrorMessage.Count == 0;
168     }
169
170     public event PropertyChangedEventHandler PropertyChanged;
171
172     private void NotifyPropertyChanged (String info)
173     {
174         if (PropertyChanged != null) {
175             PropertyChanged (this, new PropertyChangedEventArgs (
176                 info));
177         }
178     }
179
180     /// <summary>
181     /// Returns the current StockItem object as a CSV-String.
182     /// </summary>
183     /// <returns>Representation of the current object as CSV-String.</
184     returns>
185     public String CsvRepresentation ()
186     {
187         return String.Format ("{0},{1},{2},{3},{4},{5}", this.
188             StockCode, this.Name, this.SupplierName, this.UnitCost,
189             this.RequiredStock, this.CurrentStock);
190     }
191
192     /// <summary>
193     /// Attempts to create a StockItem object from a string.
194     /// </summary>
195     /// <param name="stringRepresentation">The String to be parsed to
196     StockItem.</param>
197     /// <returns>StockItem object.</returns>
198     public StockItem ParseFromString (string stringRepresentation)
199     {
200         string[] split = stringRepresentation.Split (' ','');
201         String stockCode = split[0];
202         String name = split[1];
203         String supplierName = split[2];
204         String unitCost = split[3];

```

```

197         String requiredStock = split[4];
198         String currentStock = split[5];
199         double cost = 0;
200         int reqStock = 0;
201         int currStock = 0;
202         if (!String.IsNullOrEmpty (unitCost)) {
203             cost = double.Parse (unitCost);
204         }
205         if (!String.IsNullOrEmpty (requiredStock)) {
206             reqStock = int.Parse (requiredStock);
207         }
208         if (!String.IsNullOrEmpty (currentStock)) {
209             currStock = int.Parse (currentStock);
210         }
211         return new StockItem (stockCode, name, supplierName, cost,
212                                reqStock, currStock);
213     }
214 }

```

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, IBankAccountView bankAccountView, IViewModel model)
30    {
31        this._View = view;
32        this._StockItemView = stockItemView;
33        this._BankAccountView = bankAccountView;
34        this._Model = model as AppDataManager;
35    }
36
37    /// <summary>
38    ///     <see cref="ApplicationLogic.Model.AppDataManagerClass"/>
39    /// </summary>
40    public void CreateNewStockItem ()
41    {
42        this._Model.CreateNewStockItem();
43    }
44
45    /// <summary>
46    ///     Deletes the stock item.
47    /// </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            BankAccount ba = this._View.BankAccount;
72            this._Model.DeleteBankAccount(ba);
73        }
74    }

```

```

75
76    /// <summary>
77    /// Edits the stock item.
78    /// </summary>
79    public void EditStockItem()
80    {
81    try
82    {
83        StockItem si = this._View.StockItem;
84        String stockCode = this._StockItemView.StockCode;
85        String supplier = this._StockItemView.SupplierName;
86        String name = this._StockItemView.ItemName;
87        int currentStock = this._StockItemView.CurrentStock;
88        int reqStock = this._StockItemView.RequiredStock;
89        double price = this._StockItemView.UnitCost;
90        bool areValuesValid = this._Model.ValidateStockItem(stockCode, name,
91            supplier, price, reqStock, currentStock);
92        if (areValuesValid)
93        {
94            this._Model.EditStockItem(si, stockCode, supplier, name,
95                currentStock, reqStock, price);
96        }
97        else
98        {
99            this._View.DisplayValidationErrors(this._Model.StockItemErrors());
100            this._Model.ClearStockItemErrors();
101        }
102    }
103    catch (FormatException e)
104    {
105        DisplayError(e);
106    }
107
108    /// <summary>
109    /// Edits the bank account.
110    /// </summary>
111    public void EditBankAccount()
112    {
113    try
114    {
115        BankAccount ba = this._View.BankAccount;
116        String surname = this._BankAccountView.Surname;
117        int accountNumber = this._BankAccountView.AccountNumber;
118        bool areValuesValid = this._Model.ValidateBankAccount(accountNumber,
119            surname);
120        if (areValuesValid)
121        {
122            this._Model.EditBankAccount(ba, surname, accountNumber);
123        }
124    }
125    }

```

```

123         else
124         {
125             this._View.DisplayValidationErrors(this._Model.BankAccountErrors()
126             );
127             this._Model.ClearBankAccountErrors();
128         }
129     catch (FormatException e)
130     {
131         DisplayError(e);
132     }
133
134     }
135
136     /// <summary>
137     /// Orders the item.
138     /// </summary>
139     public void OrderItem()
140     {
141         try
142         {
143             BankAccount ba = this._View.BankAccount;
144             StockItem si = this._View.StockItem;
145             this.EditStockItem();
146             int quantity = this._View.Quantity;
147             this._Model.OrderItem(si, ba, quantity);
148         }
149         catch (FormatException e)
150         {
151             DisplayError(e);
152         }
153         catch (NotEnoughFundsException e)
154         {
155             DisplayError(e);
156         }
157     }
158
159     /// <summary>
160     /// Deposits this instance.
161     /// </summary>
162     public void Deposit()
163     {
164         try
165         {
166             BankAccount ba = this._View.BankAccount;
167             double amount = this._View.Deposit;
168             this._Model.Deposit(ba, amount);
169         }
170         catch (ArgumentNullException e)
171         {
172             DisplayError(e);

```

```

173     }
174 }
175
176 /// <summary>
177 /// Withdraws this instance.
178 /// </summary>
179 public void Withdraw()
180 {
181     try
182     {
183         BankAccount ba = this._View.BankAccount;
184         double amount = this._View.Withdraw;
185         this._Model.Withdraw(ba, amount);
186     }
187     catch (ArgumentNullException e)
188     {
189         DisplayError(e);
190     }
191 }
192
193 /// <summary>
194 /// Displays the error.
195 /// </summary>
196 /// <param name="e">The e.</param>
197 private void DisplayError(Exception e)
198 {
199     ErrorMessageCollection col = new ErrorMessageCollection();
200     col.Add(new ErrorMessage(e.Message));
201     this._View.DisplayValidationErrors(col);
202 }
203
204 /// <summary>
205 /// Closes the application.
206 /// </summary>
207 public void CloseApplication()
208 {
209     if (this._View.ConfirmClose())
210         Environment.Exit(1);
211 }
212
213 /// <summary>
214 /// Loads the stock items from file.
215 /// </summary>
216 /// <param name="filePath">The file path.</param>
217 public void LoadStockItemsFromFile(String filePath)
218 {
219     this._Model.LoadStockItemsFromFile(filePath);
220 }
221
222 /// <summary>
223 /// Loads the bank accounts from file.

```

```

224     /// </summary>
225     /// <param name="filePath">The file path.</param>
226     public void LoadBankAccountsFromFile(String filePath)
227     {
228         this._Model.LoadBankAccountsFromFile(filePath);
229     }
230
231     /// <summary>
232     /// Saves the stock items to file.
233     /// </summary>
234     /// <param name="filePath">The file path.</param>
235     public void SaveStockItemsToFile(String filePath)
236     {
237         this._Model.SaveStockItemsToFile(filePath);
238     }
239
240     /// <summary>
241     /// Saves the bank accounts to file.
242     /// </summary>
243     /// <param name="filePath">The file path.</param>
244     public void SaveBankAccountsToFile(String filePath)
245     {
246         this._Model.SaveBankAccountsToFile(filePath);
247     }
248
249     /// <summary>
250     /// Sets up stock item file path.
251     /// </summary>
252     /// <param name="filePathStockItems">The file path stock items.</param>
253     public void SetUpStockItemFilePath(string filePathStockItems)
254     {
255         // TODO: Check if good.
256         this._Model.StockItemHandler.ReadFilePath = filePathStockItems;
257         this._Model.StockItemHandler.WriteFilePath = filePathStockItems;
258         this._Model.LoadStockItemsFromFile(filePathStockItems);
259     }
260
261     /// <summary>
262     /// Sets up bank accounts file path.
263     /// </summary>
264     /// <param name="filePathBankAccounts">The file path bank accounts.</param>
265     public void SetUpBankAccountsFilePath(string filePathBankAccounts)
266     {
267         // TODO: Check if good.
268         this._Model.BankAccountHandler.ReadFilePath = filePathBankAccounts;
269         this._Model.BankAccountHandler.WriteFilePath = filePathBankAccounts;
270         this._Model.LoadBankAccountsFromFile(filePathBankAccounts);
271     }
272
273     /// <summary>
274     /// Saves the bank accounts to file.

```



```

275     /// </summary>
276     public void SaveBankAccountsToFile()
277     {
278         try
279         {
280             this._Model.SaveBankAccountsToFile();
281         }
282         catch (NoFilePathSetException e)
283         {
284             DisplayError(e);
285         }
286     }
287
288     /// <summary>
289     /// Saves the stock items to file.
290     /// </summary>
291     public void SaveStockItemsToFile()
292     {
293         try
294         {
295             this._Model.SaveStockItemsToFile();
296         }
297         catch (NoFilePathSetException e)
298         {
299             DisplayError(e);
300         }
301     }
302 }
303 }

```

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(ArgumentException))]
30     public void TestWithdrawalWithTooHighValues()
31     {
32         double amountToWithdrawTooHigh = 50.0;
33         ba.Withdraw(amountToWithdrawTooHigh);
34         TestWithdrawalWithTooSmallValues();
35     }
36
37     [Test]
38     [ExpectedException(typeof(ArgumentException))]
39     public void TestWithdrawalWithTooSmallValues()
40     {
41         double amountToWithdrawTooSmall = -10.0;
42         ba.Withdraw(amountToWithdrawTooSmall);
43     }
44
45     [Test]
46     [ExpectedException(typeof(ArgumentException))]
47     public void TestDepositWithTooSmallValue()
48     {
49         double amountToDepositTooSmall = -10;
50         ba.Deposit(amountToDepositTooSmall);
51     }
52
53     [Test]
54     [ExpectedException(typeof(ArgumentException))]
55     public void TestTransferWithTooSmallValue()
56     {
57         double amountToTransferTooSmall = -10;
58         ba.Transfer(123, amountToTransferTooSmall);
59     }
60
61     [Test]
62     public void TestOrdinaryFunctions()
63     {
64         double currentValue = ba.Balance;
65         double deposit = 50.0;
66         currentValue += deposit;

```

```

67         ba.Deposit(deposit);
68         Assert.AreEqual(currentValue, ba.Balance);
69
70         double withdraw = 25.0;
71         currentValue -= withdraw;
72         ba.Withdraw(withdraw);
73         Assert.AreEqual(currentValue, ba.Balance);
74
75         double transfer = 10.50;
76         currentValue -= transfer;
77         ba.Transfer(123, transfer);
78         Assert.AreEqual(currentValue, ba.Balance);
79     }
80
81     [Test]
82     public void TestStringParsing()
83     {
84         BankAccount parseAccount = new BankAccount();
85
86         String parseOne = "123456,Rambo,500.50";
87         BankAccount ba1 = parseAccount.ParseFromString(parseOne);
88
89         String parseTwo = "000000,, ";
90         BankAccount ba2 = parseAccount.ParseFromString(parseTwo);
91     }
92
93     [Test]
94     [ExpectedException(typeof(FormatException))]
95     public void TestStringParsingInvalidValues()
96     {
97         BankAccount parseAccount = new BankAccount();
98
99         String parseOne = "abcd,Rambo,50.50";
100        BankAccount ba1 = parseAccount.ParseFromString(parseOne);
101    }
102 }
103 }

```

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     public void TestStringParsing()
62     {
63         StockItem parseItem = new StockItem();
64
65         String parseOne = "0001,Pencil Holder,John Rambo,5.50,10,15";
66
67         StockItem si = parseItem.ParseFromString(parseOne);
68         Assert.IsNotNull(si);
69
70         String parseTwo = "0001,,John Rambo,5.50,10,15";
71
72         StockItem si2 = parseItem.ParseFromString(parseTwo);
73         Assert.IsNotNull(si2);
74
75         String parseThree = "0001,,,,,";
76
77         StockItem si3 = parseItem.ParseFromString(parseThree);
78         Assert.IsNotNull(si3);
79     }
80
81     [Test]
82     [ExpectedException(typeof(FormatException))]

```

```
83     public void TestStringParsingInvalidValues()
84     {
85         StockItem parseItem = new StockItem();
86
87         String parseOne = "0001,Pencil Holder,John Rambo,abc,10,15";
88         StockItem si = parseItem.ParseFromString(parseOne);
89     }
90 }
91 }
```

Listing 24: StockItemTest.cs

BIBLIOGRAPHY

- Balzert, Helmut (2009). *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Heidelberg: Spektrum. ISBN: 9783827417053.
- 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.