SYSTEMS PROGRAMMING AND SCRIPTING

FLORIAN BERGMANN

Assessment One: Stock Manager

[October 20, 2010 at 15:47]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

CSV comma-separated values

GUI graphical user interface

MBA Management of Bank Accounts

MDA Management of Data Access

MSI Management of Stock Items

MVP model-view-presenter

TDD test-driven development

Part I

DEVELOPMENT OF A STOCK MANAGER APPLICATION

1 INTRODUCTION

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

1.1 DOCUMENT OVERVIEW

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

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

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

Chapter 6 provides an overview over the testing that has happened during the development.

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

1.2 REMIT

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

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

MSI01: Allow the management of *stock items*. Management includes the following operations: *add*, *edit*, *delete*.

MSI02: 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)

MSI03: 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*.

MSI04: 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 at least *100* items.

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

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

2 REQUIREMENT'S CHECKLIST

From the requirements stated in Section 1.2, the following were fulfilled:

MSI01: Implemented in StockItem class with getters and setters.

MSI02: Implemented in a manager-class that handles adding and deleting in-memory.

MSI03: Implemented in StockItem class.

MSI04: Implemented in AppDataManager-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 AppDataManager-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 by the GUI.

BANK ACCOUNT PERSISTENCE: It is possible to import and export bank accounts as well.

ORDER QUANTITY: It is possible to order a certain quantity instead of always ordering the required number of items.

3 DESIGN CONSIDERATIONS

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

3.1 ARCHITECTURAL OVERVIEW

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

In this pattern the presenter separates the GUI from the logical part of the application. The view communicates with the model only through the presenter. However, the model can notify the view directly of data-changes if an observer-pattern or data-binding is employed[1].:

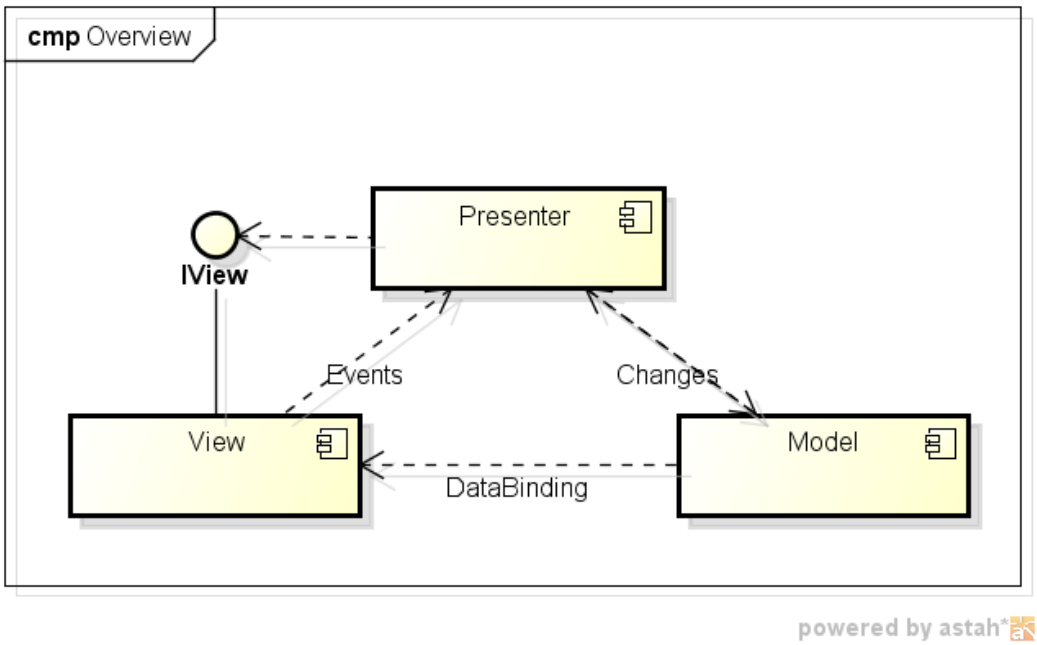

Figure 1: Architecture overview with MVP

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

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

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

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

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

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

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

3.2 USER INTERFACE

The GUI was developed completely in WinForms utilizing only standard controls provided by the .NET framework.

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

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

3.3 APPLICATION LOGIC

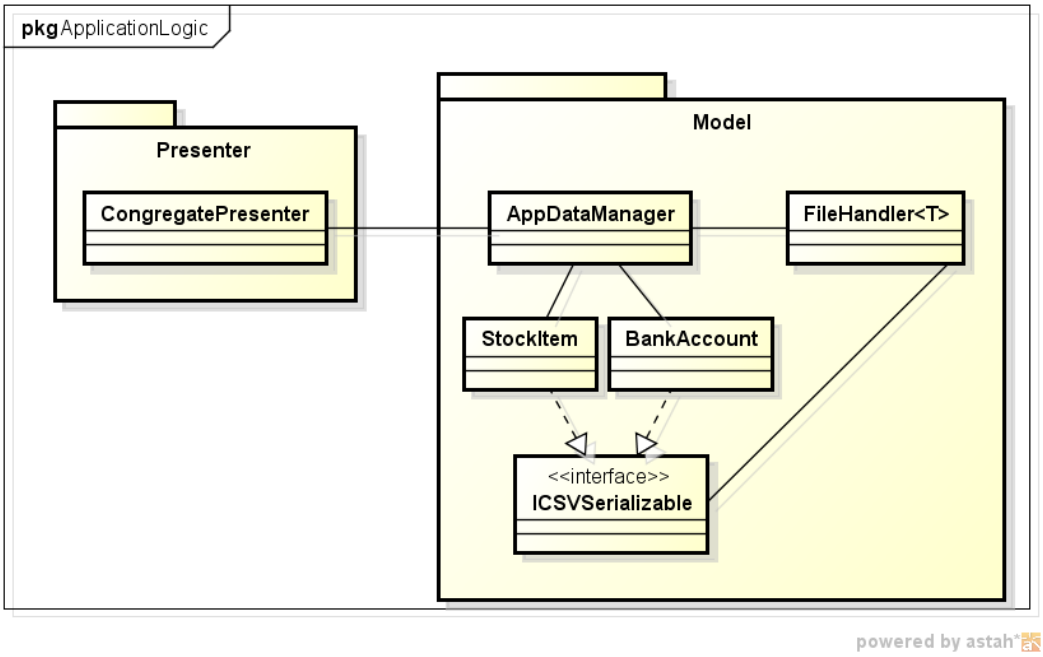

Figure 2: Abstract overview of project `application logic`

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

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

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

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

The `CongregatePresenter` seen in the picture is the connection point for the GUI part of the application and mostly forwards the requests to the `AppDataManager`.

4 USER GUIDE

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

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

4.1 MANAGE STOCK ITEMS AND BANK ACCOUNTS

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

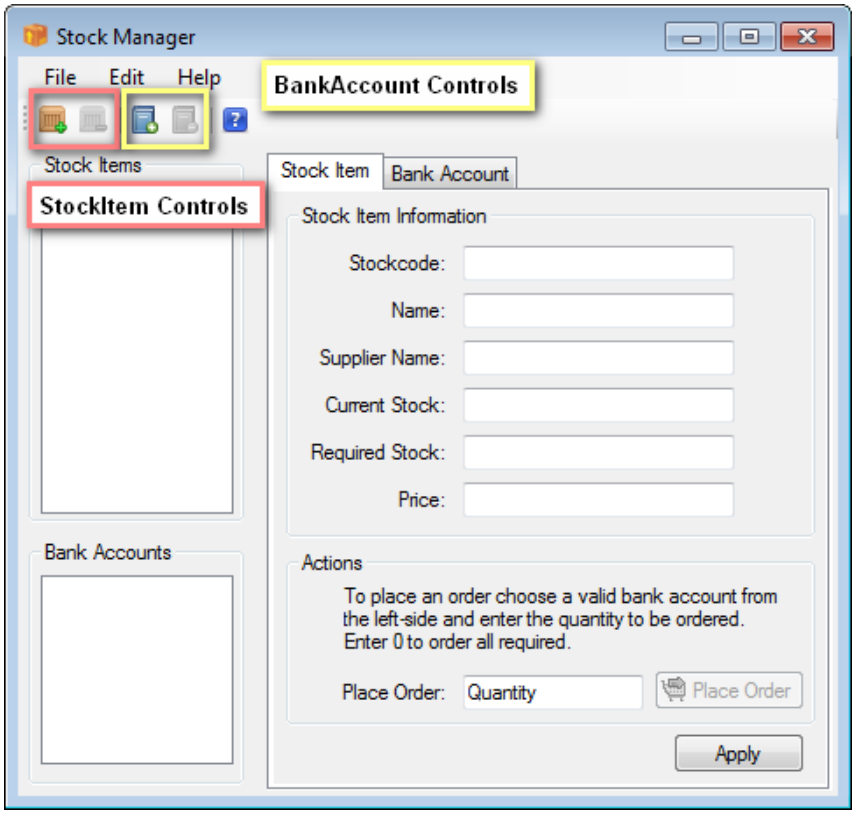

Figure 3: Main Window

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

Figure 4: Add a stock item.

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

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

Figure 5: Add a bank account.

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

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

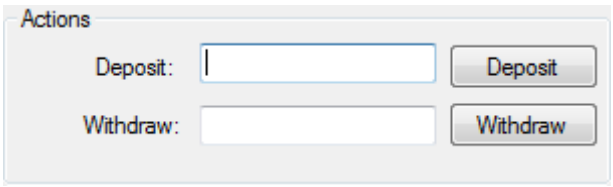

Figure 6: Depositing and withdrawing money

4.2 PLACING AN ORDER

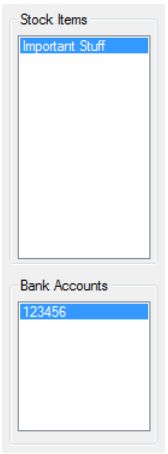

Figure 7: Selection of items.

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

Then a value 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.

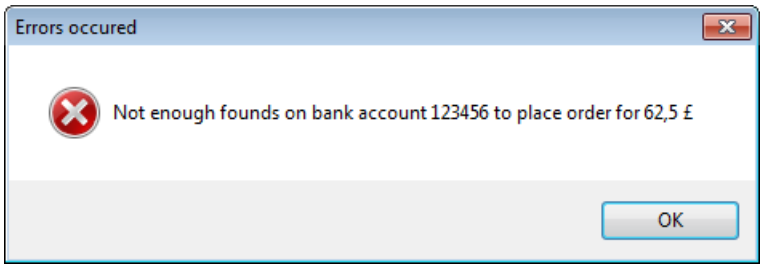

Figure 8: Placing an order without the needed funds

4.3 IMPORTING & EXPORTING DATA

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

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

4.3.1 *File menu*

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

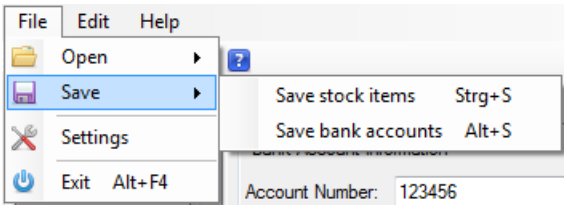

Figure 9: Saving via file menu

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` ⇒ `Settings`.

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

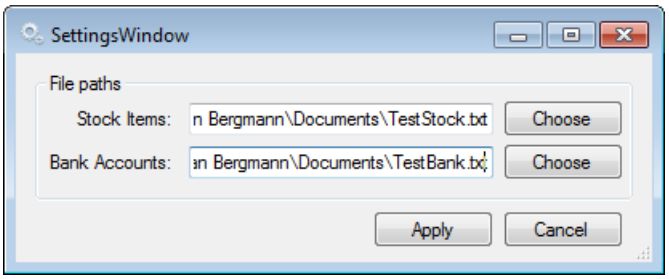

Figure 10: Settings window

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

5 DEVELOPER GUIDE

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

5.1 USERINTERFACE

The user interface package holds the WinForms representation of a possible GUI[1].

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 appearance[2].

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 datasourceupdatemode is set to "Never".
8               * This leads to the ability to enforce the use of the presenter to update
                    the values in the model.
9               * This way the validation errors can be handled by the presenter thus
                    leading to better seperation of concerns.
10              */
11             stockCodeTextBox.DataBindings.Add("Text", _Model.StockItems, "StockCode",
                   false, DataSourceUpdateMode.Never);
12             itemNameTextBox.DataBindings.Add("Text", _Model.StockItems, "Name", false,
                    DataSourceUpdateMode.Never);
13             supplierNameTextBox.DataBindings.Add("Text", _Model.StockItems, "
                   SupplierName", false, DataSourceUpdateMode.Never);
14             currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
                    ", false, DataSourceUpdateMode.Never);
15             reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
                    ", false, DataSourceUpdateMode.Never);
16             priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
                    , DataSourceUpdateMode.Never);
17
18             bankAccountsListBox.DataSource = _Model.BankAccounts;
19             bankAccountsListBox.DisplayMember = "AccountNumber";
```

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

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

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

Listing 1: Data Binding of `view` and `model`

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

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

```
1  ï»¿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  ï»¿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 view and presenter, thus allowing the possible reuse of the presenter across multiple GUIs.

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

5.2 APPLICATIONLOGIC

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

As has been pointed auto in Chapter 3, the `AppDataManager`-class works as a *facade* for the rest of the model.

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

5.2.1 *File handler*

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

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

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ApplicationLogic.Interfaces
{
    /// <summary>
    /// Used to ensure all objects will be able to persited via FileHandler class.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public interface ICSVSerializable<T>
    {
        String CsvRepresentation();

        T ParseFromString(String stringRepresentation);
    }
}
```

Listing 4: Interface `ICSVSerializable`

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

5.2.2 *Error handling*

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

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

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

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

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

Listing 5: Validate method of StockItem

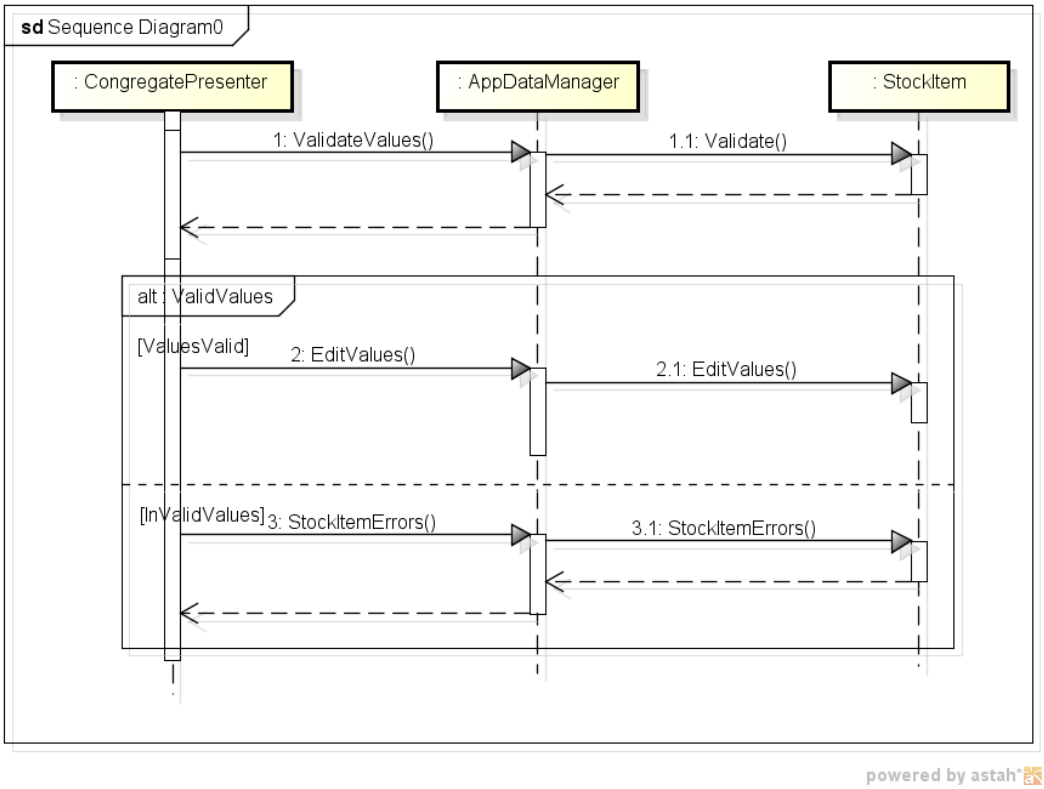

Figure 11: Sequence diagram of input validation

6 TESTING

Testing was performed in two stages:

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

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

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

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

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

The concrete error-cases that were tested are:

Under Test	Input	Output
Stock Item	Invalid StockCode	ArgumentException
Stock Item	Invalid Current Stock	ArgumentException
Stock Item	Invalid Required Stock	ArgumentException
Stock Item	Invalid Parsing	FormatException
Stock Item	Invalid Cost	ArgumentException
Bank Account	Invalid Balance	ArgumentException
Bank Account	Invalid Withdraw	ArgumentException
Bank Account	Invalid Deposit	ArgumentException
Bank Account	Invalid Transfer	ArgumentException
Bank Account	Too little funds for transfer	NotEnoughFundsException

Bank Account	Invalid Parsing	FormatException
AppDataManager	Remove invalid item	ArgumentException
AppDataManager	Perform Order	Correct sequence of statements
AppDataManager	Order with too little funds	NotEnoughFundsException
File Handler	Save without file path	NoFilePathSetException
File Handler	Load without file path	NoFilePathSetException

Table 1: Performed tests.

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

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

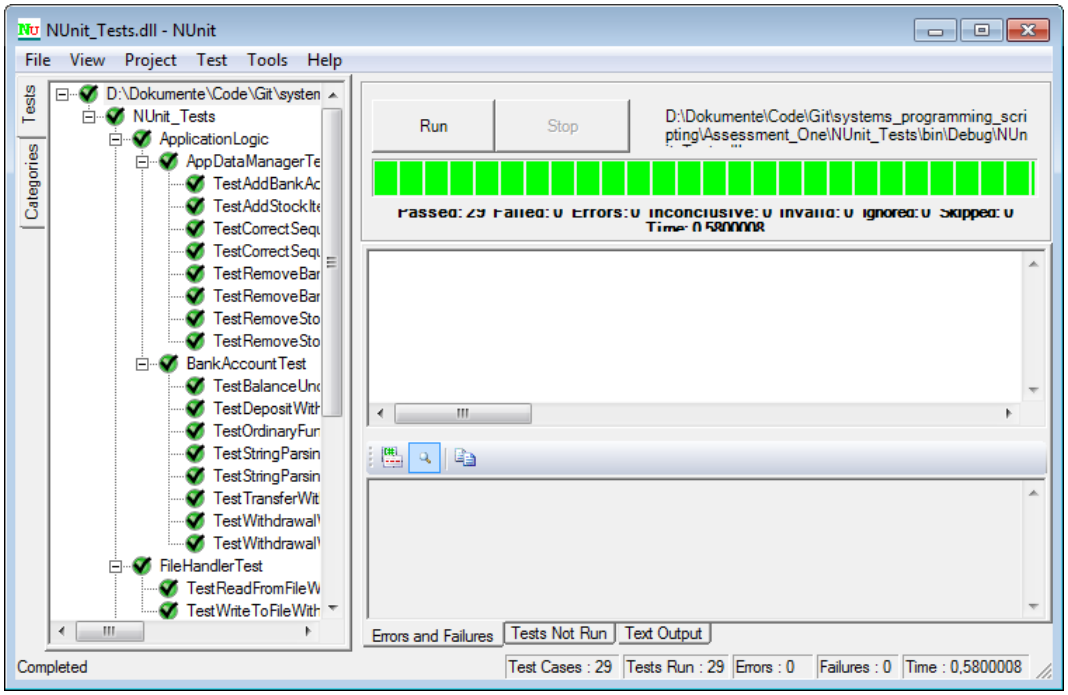

Figure 12: NUnit test case run

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

7 CONCLUSIONS

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

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

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

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

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

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

Part II

APPENDIX

APPENDIX: SOURCE CODE

A.1 VIEW

```
ï»¿using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using ApplicationLogic.Interfaces;
using ApplicationLogic.Presenter;
using ApplicationLogic.Model;
using System.Collections.Specialized;
using Assessment_One.Properties;

namespace Assessment_One
{
    public partial class MainWindow : Form, ICongregateView, IStockItemView,
        IBankAccountView
    {
        private CongregatePresenter _Presenter;
        private IViewModel _Model;
        private List<Control> backgroundColorChanged;

        private const int STOCKITEMTAB = 0;
        private const int BANKACCOUNTTAB = 1;

        /// <summary>
        /// Initializes a new instance of the <see cref="MainWindow"/> class.
        /// </summary>
        public MainWindow()
        {
            InitializeComponent();
            this._Model = new AppDataManager();
            _Presenter = new CongregatePresenter(this, this, this, this._Model);
            this.backgroundColorChanged = new List<Control>();
            LoadFilePathSettings();
            SetUpDataBindings();
        }

```

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

```
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
                e)
101         {
102             if (this.openFileDialog.ShowDialog() == DialogResult.OK)
103             {
104                 OpenFileDialog file = this.openFileDialog;
105                 this._Presenter.LoadBankAccountsFromFile(file.FileName);
106             }
107         }
108
109         private void saveStockItemsToolStripMenuItem_Click(object sender, EventArgs e)
110         {
111             if (this.saveFileDialog.ShowDialog() == DialogResult.OK)
112             {
113                 SaveFileDialog file = this.saveFileDialog;
114                 this._Presenter.SaveStockItemsToFile(file.FileName);
115             }
116         }
117
118         private void saveBankAccountsToolStripMenuItem_Click(object sender, EventArgs
                e)
119         {
120             if (this.saveFileDialog.ShowDialog() == DialogResult.OK)
121             {
122                 SaveFileDialog file = this.saveFileDialog;
123                 this._Presenter.SaveBankAccountsToFile(file.FileName);
124             }
125         }
126
127         private void applyButton_Click(object sender, EventArgs e)
128         {
129             this.ResetColoring();
130             this._Presenter.EditStockItem();
131         }
132
133         private void ResetColoring()
134         {
135             foreach (Control control in this.backgroundColorChanged)
136             {
```

```
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 occured",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
429         }
430 
431         /// <summary>
432         /// Asks fir confirmation of a deletion.
433         /// </summary>
434         /// <returns></returns>
435         public bool ConfirmDelete()
436         {
437             DialogResult result = MessageBox.Show("Are you sure you want to delete
                    this item?", "Confirm delete", MessageBoxButtons.YesNo, MessageBoxIcon
                    .Question);
438             return result == DialogResult.Yes;
439         }
```

```
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 datasourceupdatemode is set to "Never".
504              * This leads to the ability to enforce the use of the presenter to update
                       the values in the model.
505              * This way the validation errors can be handled by the presenter thus
                     leading to better seperation of concerns.
506              */
507             stockCodeTextBox.DataBindings.Add("Text", _Model.StockItems, "StockCode",
                     false, DataSourceUpdateMode.Never);
508             itemNameTextBox.DataBindings.Add("Text", _Model.StockItems, "Name", false,
                      DataSourceUpdateMode.Never);
509             supplierNameTextBox.DataBindings.Add("Text", _Model.StockItems, "
                    SupplierName", false, DataSourceUpdateMode.Never);
510             currStockTextBox.DataBindings.Add("Text", _Model.StockItems, "CurrentStock
                    ", false, DataSourceUpdateMode.Never);
511             reqStockTextBox.DataBindings.Add("Text", _Model.StockItems, "RequiredStock
                    ", false, DataSourceUpdateMode.Never);
512             priceTextBox.DataBindings.Add("Text", _Model.StockItems, "UnitCost", false
                    , DataSourceUpdateMode.Never);
513 
514             bankAccountsListBox.DataSource = _Model.BankAccounts;
515             bankAccountsListBox.DisplayMember = "AccountNumber";
516 
517             accountNumberTextBox.DataBindings.Add("Text", _Model.BankAccounts, "
                    AccountNumber", false, DataSourceUpdateMode.Never);
518             nameTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Surname", false
                    , DataSourceUpdateMode.Never);
519             balanceTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Balance",
                     false, DataSourceUpdateMode.Never);
520         }
521 
522         private void quantityTextBox_TextChanged(object sender, EventArgs e)
523         {
524             String newText = quantityTextBox.Text;
525             int parseInt = 0;
526             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  ï»¿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
                bankAccountsFilePath)
58          {
59              Settings settings = Settings.Default;
60              settings.StockItemFilePath = stockItemsFilePath;
61              settings.BankAccountFilePath = bankAccountsFilePath;
62              settings.Save();
63          }
64      }
65  }
```

Listing 7: Settings.cs

A.2 APPLICATION LOGIC

A.2.1 *Interfaces Package*

```
1   ï»¿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   ï»¿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: ICongegrateView.cs

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

Listing 11: IStockItemView.cs

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

Listing 12: IViewModel.cs

A.2.2 *Model Package*

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

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

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

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

```
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
                quantity will be ordered.
213         /// Otherwise the provided quantity will be ordered.
214         /// </summary>
215         /// <param name="indexStockItem">Index of the stock item in the stock items
                list.</param>
216         /// <param name="indexBankAccount">Index of the bank account in the bank
                account list.</param>
217         /// <param name="quantity">The quantity to be ordered. 0 orders the required
                quantity.</param>
218         public void OrderItem(StockItem si, BankAccount ba, int quantity)
219         {
220             if (ba != null && si != null)
221             {
222                 bool buyExcessStock = false;
223                 if (quantity == 0)
224                 {
225                     quantity = si.RequiredStock;
226                 }
227                 else
228                 {
229                     buyExcessStock = true;
230                 }
231                 double priceOfOrder = quantity * si.UnitCost;
232                 if (priceOfOrder < ba.Balance)
233                 {
234                     ba.Transfer(1, priceOfOrder);
235                     si.CurrentStock += quantity;
236                     /*
237                      * Allow the user to buy more than needed.
238                      */
239                     if (buyExcessStock)
240                     {
241                         if (quantity > si.RequiredStock)
242                         {
243                             si.RequiredStock = 0;
```

```
244                         }
245                     }
246                 }
247                 else
248                 {
249                     throw new NotEnoughFundsException(String.Format("Not enough founds
                           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             this.StockItems.Clear();
302             foreach (StockItem item in stockItems)
303             {
304                 this.StockItems.Add(item);
305             }
306         }
307
308         /// <summary>
309         /// Will reload the BankAccount collection from the specified file.
310         /// Will overwrite the currently existing BankAccount collection.
311         /// </summary>
312         /// <param name="filePath">The path to the file</param>
313         internal void LoadBankAccountsFromFile(string filePath)
314         {
315             this.BankAccountHandler.ReadFilePath = filePath;
316             IList<BankAccount> bankAccounts = BankAccountHandler.LoadFromFile(new
                    BankAccount());
317             this.BankAccounts.Clear();
318             foreach (BankAccount item in bankAccounts)
319             {
320                 this.BankAccounts.Add(item);
321             }
322         }
323
324         /// <summary>
325         /// Will save the current StockItem collection to the specified file.
326         /// </summary>
327         /// <param name="filePath">The path to the file</param>
328         internal void SaveStockItemsToFile(string filePath)
329         {
330             this.StockItemHandler.WriteFilePath = filePath;
331             this.StockItemHandler.SaveToFile(this.StockItems);
332         }
333
334         /// <summary>
335         /// Will save the current BankAccount collection to the specified file.
336         /// </summary>
337         /// <param name="filePath">The path to the file</param>
338         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
                FileHandler.
346         /// Throws NoFilePathSetException if no path is set.
347         /// </summary>
348         internal void SaveBankAccountsToFile()
349         {
350             this.BankAccountHandler.SaveToFile(this.BankAccounts);
351         }
352 
353         /// <summary>
354         /// Will save the current BankAccount collection to the file stored in the
                FileHandler.
355         /// Throws NoFilePathSetException if no path is set.
356         /// </summary>
357         internal void SaveStockItemsToFile()
358         {
359             this.StockItemHandler.SaveToFile(this.StockItems);
360         }
361     }
362 }
```

Listing 13: AppDataManager.cs

```
1  ï»¿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<
               BankAccount>
14         {
15                 private int _AccountNumber;
16                 /// <summary>
17                 /// Gets or sets the account number.
18                 /// </summary>
19                 /// <value>The account number.</value>
20                 public virtual int AccountNumber {
21                         get { return _AccountNumber; }
```

```
22                      set {
23                              _AccountNumber = value;
24                              this.NotifyPropertyChanged ("AccountNumber");
25                      }
26              }
27
28              private String _Surname;
29              /// <summary>
30              /// Gets or sets the surname.
31              /// </summary>
32              /// <value>The surname.</value>
33              public virtual String Surname {
34                      get { return _Surname; }
35                      set {
36                              _Surname = value;
37                              this.NotifyPropertyChanged ("Surname");
38                      }
39              }
40
41              private double _Balance;
42              /// <summary>
43              /// Gets or sets the balance.
44              /// </summary>
45              /// <value>The balance.</value>
46              public virtual double Balance {
47                      get { return _Balance; }
48                      private set {
49                              if (value < 0.0) {
50                                      throw new ArgumentException ("This class does
                                            not allow a balance smaller than 0.");
51                              } else {
52                                      _Balance = value;
53                                      this.NotifyPropertyChanged ("Balance");
54                              }
55
56                      }
57              }
58
59              public static ErrorMessageCollection ErrorMessages = new
                    ErrorMessageCollection ();
60
61              /// <summary>
62              /// Initializes a new instance of the <see cref="BankAccount"/> class.
63              /// </summary>
64              public BankAccount ()
65              {
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
                    0.</param>
86              public void Withdraw (double amount)
87              {
88                      if (amount > 0.0) {
89                              if (Balance > amount) {
90                                      this.Balance -= amount;
91                              } else {
92                                      throw new ArgumentException ("Not enough
                                            founds on bank account to withdraw.");
93                              }
94                      } else {
95                              throw new ArgumentException ("There are not enough
                                    funds present to fulfill the required action.");
96                      }
97              }
98
99              /// <summary>
100             /// Allows the deposit of money to this account.
101             /// </summary>
102             /// <param name="amount">Amount to be deposited - must be greater than
                    0.</param>
103             public void Deposit (double amount)
104             {
105                     if (amount <= 0.0) {
106                             throw new ArgumentException ("To withdraw money please
                                    use the appropriate function.");
107                     } else {
108                             this.Balance += amount;
109                     }
110             }
111
112             /// <summary>
113             /// Amount to be transfered to another account.
114             /// NOTE: This method is a fake to simulate "real" banking. The money
                    will not be transfered to any account.
115             /// </summary>
```

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

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

Listing 14: BankAccount.cs

```
1 ï»¿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  ï»¿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 errormessages 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

```
using System;
using System.Collections;
using System.Collections.Generic;
using ApplicationLogic.Interfaces;
using System.IO;
using ApplicationLogic.Model;

namespace ApplicationLogic
{
    /// <summary>
    /// Handles reading and writing from files.
    /// </summary>
    public class FileHandler<T> where T : ICSVSerializable<T>
    {

        public String ReadFilePath;
        public String WriteFilePath;

        public FileHandler()
        {
        }

        public FileHandler(String readFilePath, String writeFilePath)
        {
            this.ReadFilePath = readFilePath;
            this.WriteFilePath = writeFilePath;
        }

        /// <summary>
        /// Attempts to write the specified collection to the file specified in the
            Handler.
        /// Throws NoFilePathSetException if no file has yet been set.
        /// </summary>
        /// <param name="elements">Collection to be saved to file.</param>
        public void SaveToFile(IList<T> elements)
        {
            if (!String.IsNullOrEmpty(this.WriteFilePath))
            {
                FileStream writeFile = File.Open(WriteFilePath, FileMode.Create);
                using (StreamWriter sw = new StreamWriter(writeFile))
                {
                    foreach (T item in elements)
                    {
                        sw.Write(item.CsvRepresentation());
                        sw.Write("\n");
                    }
                }
```

```
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(
                                                                readString);
74                                      returnList.Add(t);
75                                                  } catch (FormatException ex) {
76                                                          Console.WriteLine(ex.
                                                                StackTrace);
77                                                  }
78                      }
79                  }
80                  return returnList;
81              }
82              else
83              {
84                  throw new NoFilePathSetException("No file path to write to set.");
85              }
86          }
87      }
88  }
```

Listing 17: FileHandler.cs

```
1  ï»¿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  ï»¿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  ï»¿using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Text.RegularExpressions;
6  using System.ComponentModel;
7  using ApplicationLogic.Interfaces;
8
9  namespace ApplicationLogic.Model
10 {
11         /// <summary>
12         /// Stores all necessary data for a StockItem.
13         /// </summary>
14         public class StockItem : INotifyPropertyChanged, ICSVSerializable<StockItem>
15         {
16         private const string REGEX = "^[0-9]{4}$";
17                 private String _StockCode;
18                 public virtual String StockCode {
```

```
19                      get { return _StockCode; }
20                      set {
21                              if (!IsValidStockCode (value)) {
22                                      throw new ArgumentException ("Provided
                                            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 virtual String Name {
32                      get { return _Name; }
33                      set {
34                              _Name = value;
35                              this.NotifyPropertyChanged ("Name");
36                      }
37              }
38
39              private String _SupplierName;
40              public virtual String SupplierName {
41                      get { return _SupplierName; }
42                      set {
43                              _SupplierName = value;
44                              this.NotifyPropertyChanged ("SupplierName");
45                      }
46              }
47
48              private double _UnitCost;
49              public virtual 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 virtual int RequiredStock {
64                      get { return _RequiredStock; }
65                      set {
66                              if (value < 0) {
```

```
67                                  throw new ArgumentException ("Can not require
                                        less than 0 items.");
68                          } else {
69                                  _RequiredStock = value;
70                                  this.NotifyPropertyChanged ("RequiredStock");
71                          }
72
73                  }
74          }
75
76          private int _CurrentStock;
77          public virtual int CurrentStock {
78                  get { return _CurrentStock; }
79                  set {
80                          if (value < 0) {
81                                  throw new ArgumentException ("Current stock
                                        can not be less than 0 items.");
82                          } else {
83                                  _CurrentStock = value;
84                                  this.NotifyPropertyChanged ("CurrentStock");
85                          }
86
87                  }
88          }
89
90          public static ErrorMessageCollection ErrorMessages = new
                ErrorMessageCollection ();
91
92          public StockItem ()
93          {
94          }
95
96          public StockItem (String stockCode, String name, String supplierName,
                double unitCost, int required, int currentStock)
97          {
98                  this.StockCode = stockCode;
99                  this.Name = name;
100                 this.SupplierName = supplierName;
101                 this.UnitCost = unitCost;
102                 this.RequiredStock = required;
103                 this.CurrentStock = currentStock;
104         }
105
106         /// <summary>
107         /// Checks if the stock code conforms to a certain format.
108         /// Current format is exactly four numbers, with leading 0 allowed.
109         /// </summary>
110         /// <param name="value">The string that must be checked against the
                schema.</param>
111         /// <returns>True if the string conforms to the schema, false
                otherwise.</returns>
```

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

```
154                     }
155                     if (unitCost < 0.0) {
156                             ErrorMessages.Add (new ErrorMessage ("Unit costs must
                                    be greater or equal 0."));
157                     }
158                     if (required < 0) {
159                             ErrorMessages.Add (new ErrorMessage ("Required must be
                                     greater or equal 0."));
160                     }
161                     if (currentStock < 0) {
162                             ErrorMessages.Add (new ErrorMessage ("Current must be
                                    greater or equal 0."));
163                     }
164                     return ErrorMessages.Count == 0;
165             }
166
167             public event PropertyChangedEventHandler PropertyChanged;
168
169             private void NotifyPropertyChanged (String info)
170             {
171                     if (PropertyChanged != null) {
172                             PropertyChanged (this, new PropertyChangedEventArgs (
                                    info));
173                     }
174             }
175
176             /// <summary>
177             /// Returns the current StockItem object as a CSV-String.
178             /// </summary>
179             /// <returns>Representation of the current object as CSV-String.</
                    returns>
180             public String CsvRepresentation ()
181             {
182                     return String.Format ("{0},{1},{2},{3},{4},{5}", this.
                            StockCode, this.Name, this.SupplierName, this.UnitCost,
                            this.RequiredStock, this.CurrentStock);
183             }
184
185             /// <summary>
186             /// Attempts to create a StockItem object from a string.
187             /// </summary>
188             /// <param name="stringRepresentation">The String to be parsed to
                    StockItem.</param>
189             /// <returns>StockItem object.</returns>
190             public StockItem ParseFromString (string stringRepresentation)
191             {
192                     string[] split = stringRepresentation.Split (',');
193                     String stockCode = split[0];
194                     String name = split[1];
195                     String supplierName = split[2];
196                     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,
                                reqStock, currStock);
212                 }
213         }
214 }
```

Listing 20: StockItem.cs

A.2.3 *Presenter Package*

```
1  ï»¿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
           application logic.
13     /// </summary>
14         public class CongregatePresenter
15         {
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,
                        supplier, price, reqStock, currentStock);
91                  if (areValuesValid)
92                  {
93                      this._Model.EditStockItem(si, stockCode, supplier, name,
                            currentStock, reqStock, price);
94                  }
95                  else
96                  {
97                      this._View.DisplayValidationErrors(this._Model.StockItemErrors());
98                      this._Model.ClearStockItemErrors();
99                  }
100             }
101             catch (FormatException e)
102             {
103                 DisplayError(e);
104             }
105
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,
                        surname);
119                 if (areValuesValid)
120                 {
121                     this._Model.EditBankAccount(ba, surname, accountNumber);
122                 }
```

```
123                 else
124                 {
125                     this._View.DisplayValidationErrors(this._Model.BankAccountErrors()
                            );
126                     this._Model.ClearBankAccountErrors();
127                 }
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  ï»¿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   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using ApplicationLogic.Model;
6   using NUnit.Framework;
7
8   namespace NUnit_Tests
9   {
10      [TestFixture]
11      public class StockItemTest
12      {
13          private StockItem si;
14
15          [SetUp]
16          public void SetUp()
17          {
18              si = new StockItem("1234", "Test", "Test", 10.0, 5, 5);
19          }
20
21          [Test]
22          [ExpectedException(typeof(ArgumentException))]
23          public void TestLessThan0Cost()
24          {
25              si = new StockItem("1234", "Test", "Test", -1.0, 5, 5);
26          }
27
28          [Test]
29          [ExpectedException(typeof(ArgumentException))]
30          public void TestLessThan0CurrentStock()
31          {
```

```
32              si = new StockItem("1234", "Test", "Test", 1.0, -1, 5);
33          }
34
35          [Test]
36          [ExpectedException(typeof(ArgumentException))]
37          public void TestLessThan0RequiredStock()
38          {
39              si = new StockItem("1234", "Test", "Test", 1.0, 5, -1);
40          }
41
42          [Test]
43          public void TestIsValidStockCode()
44          {
45              bool validSC = StockItem.IsValidStockCode("1234");
46              Assert.IsTrue(validSC);
47              bool tooLongSC = StockItem.IsValidStockCode("123456");
48              Assert.IsFalse(tooLongSC);
49              bool stringSC = StockItem.IsValidStockCode("test");
50              Assert.IsFalse(stringSC);
51          }
52
53          [Test]
54          [ExpectedException(typeof(ArgumentNullException))]
55          public void TestIsValidStockCodeRaiseException()
56          {
57              StockItem.IsValidStockCode(null);
58          }
59
60          [Test]
61          [ExpectedException(typeof(ArgumentException))]
62          public void TestCreateStockItemInvalidStockCode()
63          {
64              String invalidStockCode = "00001";
65              StockItem si = new StockItem(invalidStockCode, "", "", 0.0, 0, 0);
66          }
67
68          [Test]
69          [ExpectedException(typeof(ArgumentException))]
70          public void TestCreateStockItemInvalidCost()
71          {
72              double invalidCost = -1.0;
73              StockItem si = new StockItem("0001", "", "", invalidCost, 0, 0);
74          }
75
76          [Test]
77          [ExpectedException(typeof(ArgumentException))]
78          public void TestCreateStockItemInvalidRequiredStock()
79          {
80              int invalidStock = -1;
81              StockItem si = new StockItem("0001", "", "", 0.0, invalidStock, 0);
82          }
```

```
83
84          [Test]
85          [ExpectedException(typeof(ArgumentException))]
86          public void TestCreateStockItemInvalidCurrentStock()
87          {
88              int invalidStock = -1;
89              StockItem si = new StockItem("0001", "", "", 0.0,  0, invalidStock);
90          }
91
92                  [Test]
93                  public void TestStringParsing()
94                  {
95                          StockItem parseItem = new StockItem();
96
97                          String parseOne = "0001,Pencil Holder,John Rambo,5.50,10,15";
98
99                          StockItem si = parseItem.ParseFromString(parseOne);
100                         Assert.IsNotNull(si);
101
102                         String parseTwo = "0001,,John Rambo,5.50,10,15";
103
104                         StockItem si2 = parseItem.ParseFromString(parseTwo);
105                         Assert.IsNotNull(si2);
106
107                         String parseThree = "0001,,,,,";
108
109                         StockItem si3 = parseItem.ParseFromString(parseThree);
110                         Assert.IsNotNull(si3);
111                 }
112
113                 [Test]
114                 [ExpectedException(typeof(FormatException))]
115                 public void TestStringParsingInvalidValues()
116                 {
117                         StockItem parseItem = new StockItem();
118
119                         String parseOne = "0001,Pencil Holder,John Rambo,abc,10,15";
120                         StockItem si = parseItem.ParseFromString(parseOne);
121                 }
122     }
123 }
```

Listing 24: StockItemTest.cs

```
1 ï»¿using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using NUnit.Framework;
6 using Moq;
7 using ApplicationLogic.Model;
```

```
8
9  namespace NUnit_Tests.ApplicationLogic
10 {
11     [TestFixture]
12     public class AppDataManagerTest
13     {
14         private AppDataManager _Manager;
15         private StockItem _Stock;
16         private BankAccount _Account;
17
18         [SetUp]
19         public void SetUp()
20         {
21             this._Manager = new AppDataManager();
22             this._Stock = new StockItem();
23             this._Account = new BankAccount();
24         }
25
26         [Test]
27         public void TestAddStockItem()
28         {
29             this._Manager.CreateNewStockItem();
30             Assert.AreEqual(1, this._Manager.StockItems.Count);
31         }
32
33         [Test]
34         public void TestAddBankAccount()
35         {
36             this._Manager.CreateNewBankAccount();
37             Assert.AreEqual(1, this._Manager.BankAccounts.Count);
38         }
39
40         [Test]
41         public void TestRemoveStockItem()
42         {
43             this._Manager.CreateNewStockItem();
44             Assert.AreEqual(1, this._Manager.StockItems.Count);
45             StockItem remove = this._Manager.StockItems.ElementAt(0);
46             this._Manager.DeleteStockItem(remove);
47             Assert.AreEqual(0, this._Manager.StockItems.Count);
48         }
49
50         [Test]
51         public void TestRemoveBankAccount()
52         {
53             this._Manager.CreateNewBankAccount();
54             Assert.AreEqual(1, this._Manager.BankAccounts.Count);
55             BankAccount remove = this._Manager.BankAccounts.ElementAt(0);
56             this._Manager.DeleteBankAccount(remove);
57             Assert.AreEqual(0, this._Manager.BankAccounts.Count);
58         }
```

```
59
60          [Test]
61          [ExpectedException(typeof(ArgumentException))]
62          public void TestRemoveStockItemNotPresent()
63          {
64              StockItem si = new StockItem();
65              this._Manager.DeleteStockItem(si);
66          }
67
68          [Test]
69          [ExpectedException(typeof(ArgumentException))]
70          public void TestRemoveBankAccountNotPresent()
71          {
72              BankAccount ba = new BankAccount();
73              this._Manager.DeleteBankAccount(ba);
74          }
75
76          [Test]
77          public void TestCorrectSequenceOfOrdering()
78          {
79              var mockBa = new Mock<BankAccount>();
80              var mockSi = new Mock<StockItem>();
81
82              mockBa.Setup(ba => ba.Balance).Returns(50.0);
83
84              this._Manager.BankAccounts.Add(mockBa.Object);
85              this._Manager.StockItems.Add(mockSi.Object);
86              this._Manager.OrderItem(mockSi.Object, mockBa.Object, 0);
87
88              mockBa.VerifyGet(ba => ba.Balance);
89              mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.AtMostOnce());
90          }
91
92          [Test]
93          [ExpectedException(typeof(NotEnoughFundsException))]
94          public void TestCorrectSequenceOnInvalidFunds()
95          {
96              var mockBa = new Mock<BankAccount>();
97              var mockSi = new Mock<StockItem>();
98
99              mockBa.Setup(ba => ba.Balance).Returns(0.0);
100             mockSi.Setup(si => si.UnitCost).Returns(10.0);
101
102             this._Manager.BankAccounts.Add(mockBa.Object);
103             this._Manager.StockItems.Add(mockSi.Object);
104             this._Manager.OrderItem(mockSi.Object, mockBa.Object, 0);
105
106             mockBa.VerifyGet(ba => ba.Balance);
107             mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.Never());
108         }
109     }
```

```
110 }
```

Listing 25: AppDataManagerTest.cs

BIBLIOGRAPHY

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

Boodhoo, Jean-Paul (2006). *Design Patterns: Model View Presenter*. English. Microsoft. URL: http://msdn.microsoft.com/en-us/magazine/cc188690.aspx (visited on 19/10/2010).

Dorman, Scott (2010). *Sams Teach Yourself Visual CSharp®2010 in 24 Hours: Complete Starter Kit*. Sams Publishing. ISBN: 978-0-672-33101-5.

Freeman, Eric and Elisabeth Freeman (2004). *Head First - Design Pattern*. Ed. by Mike Loukides. O'Reilly. ISBN: 0-596-00712-4.

Noyes, Brian (2006). *Data Binding With Windows Forms 2.0 - Programming Smart Client Data Applications With .NET*. Addison Wesley Professional. ISBN: 978-0-321-26892-1.

Sommerville, Ian (2006). *Software Engineering*. 8th ed. Addison Wesley. ISBN: 9780321210265.

Stellman, Andrew and Jennifer Greene (2010). *Head First CSharp*. O'Reilly. ISBN: 978-1-449-38034-2.