

# SYSTEMS PROGRAMMING AND SCRIPTING

FLORIAN BERGMANN

Assessment One: Stock Manager

## CONTENTS

---

<b>I DEVELOPMENT OF A STOCK MANAGER APPLICATION</b>	<b>1</b>
<b>1 INTRODUCTION</b>	<b>2</b>
1.1 Document overview . . . . .	2
1.2 Remit . . . . .	2
<b>2 REQUIREMENT'S CHECKLIST</b>	<b>4</b>
<b>3 DESIGN CONSIDERATIONS</b>	<b>6</b>
3.1 Architectural overview . . . . .	6
3.2 User Interface . . . . .	7
3.3 Application Logic . . . . .	8
<b>4 USER GUIDE</b>	<b>9</b>
4.1 Manage stock items and bank accounts . . . . .	9
4.2 Placing an order . . . . .	10
4.3 Importing & exporting data . . . . .	11
4.3.1 File menu . . . . .	11
4.3.2 Menu-bar icon . . . . .	11
<b>5 DEVELOPER GUIDE</b>	<b>13</b>
5.1 UserInterface . . . . .	13
5.2 ApplicationLogic . . . . .	15
5.2.1 File handler . . . . .	16
5.2.2 Error handling . . . . .	16
<b>6 TESTING</b>	<b>19</b>
<b>7 CONCLUSIONS</b>	<b>22</b>
<b>II APPENDIX</b>	<b>23</b>
<b>A APPENDIX: SOURCE CODE</b>	<b>24</b>
A.1 View . . . . .	24
A.2 Application Logic . . . . .	36
A.2.1 Interfaces Package . . . . .	36
A.2.2 Model Package . . . . .	39
A.2.3 Presenter Package . . . . .	59
A.3 Tests . . . . .	66
<b>BIBLIOGRAPHY</b>	<b>75</b>

## LIST OF FIGURES

---

Figure 1	Architecture overview with model-view-presenter (MVP) . . .	6
Figure 2	Abstract overview of project application logic . . . . .	8
Figure 3	Main Window . . . . .	9
Figure 4	Add a stock item. . . . .	10
Figure 5	Add a bank account. . . . .	10
Figure 6	Depositing and withdrawing money . . . . .	10
Figure 7	Selection of items. . . . .	10
Figure 8	Placing an order without the needed funds . . . . .	11
Figure 9	Saving via file menu . . . . .	11
Figure 10	Settings window . . . . .	12
Figure 11	Sequence diagram of input validation . . . . .	18
Figure 12	NUnit test case run . . . . .	21

## LIST OF TABLES

---

Table 1	Performed tests. . . . .	20
---------	--------------------------	----

## LISTINGS

---

Listing 1	Data Binding of view and model . . . . .	13
Listing 2	Example interface IStockItemView . . . . .	14
Listing 3	Interface ICongregateView . . . . .	14
Listing 4	Interface ICSVSerializable . . . . .	16
Listing 5	Validate method of StockItem . . . . .	16
Listing 6	MainWindow.cs . . . . .	24

Listing 7	Settings.cs . . . . .	35
Listing 8	IBankAccountView.cs . . . . .	36
Listing 9	ICongregateView.cs . . . . .	37
Listing 10	ICSVSerializable.cs . . . . .	38
Listing 11	IStockItemView.cs . . . . .	38
Listing 12	IViewModel.cs . . . . .	38
Listing 13	AppDataManager.cs . . . . .	39
Listing 14	BankAccount.cs . . . . .	46
Listing 15	ErrorMessage.cs . . . . .	51
Listing 16	ErrorMessageCollection.cs . . . . .	51
Listing 17	FileHandler.cs . . . . .	52
Listing 18	NoFilePathSetException.cs . . . . .	54
Listing 19	NotEnoughFundsException.cs . . . . .	54
Listing 20	StockItem.cs . . . . .	54
Listing 21	CongregatePresenter.cs . . . . .	59
Listing 22	BankAccountTest.cs . . . . .	66
Listing 23	FileHandlerTest.cs . . . . .	69
Listing 24	StockItemTest.cs . . . . .	69
Listing 25	AppDataManagerTest.cs . . . . .	72

## ACRONYMS

---

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

## Part I

# DEVELOPMENT OF A STOCK MANAGER APPLICATION

## INTRODUCTION

---

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

### 1.1 DOCUMENT OVERVIEW

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

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

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

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

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

### 1.2 REMIT

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

The requirements, as understood by the contractor, are as follows <sup>1</sup>:

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

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

---

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

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

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

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

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

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

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

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

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

MDA04: The ordering of the files is as follows:

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

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

MDA05: The application should be able to handle 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.

## REQUIREMENT'S CHECKLIST

---

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

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

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

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



## DESIGN CONSIDERATIONS

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

### 3.1 ARCHITECTURAL OVERVIEW

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

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

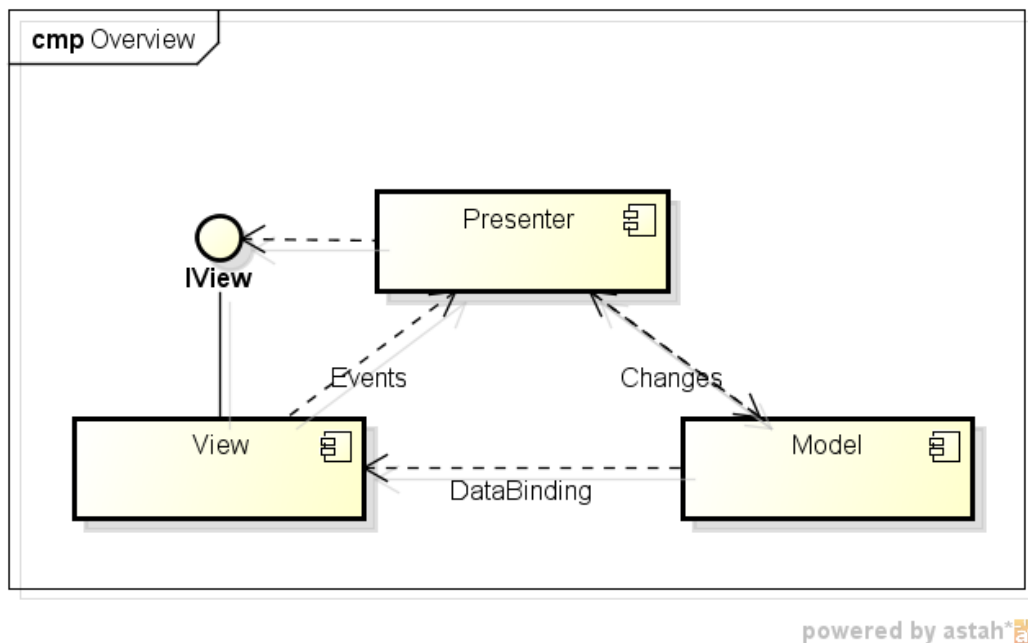


Figure 1: Architecture overview with MVP

<sup>1</sup> 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

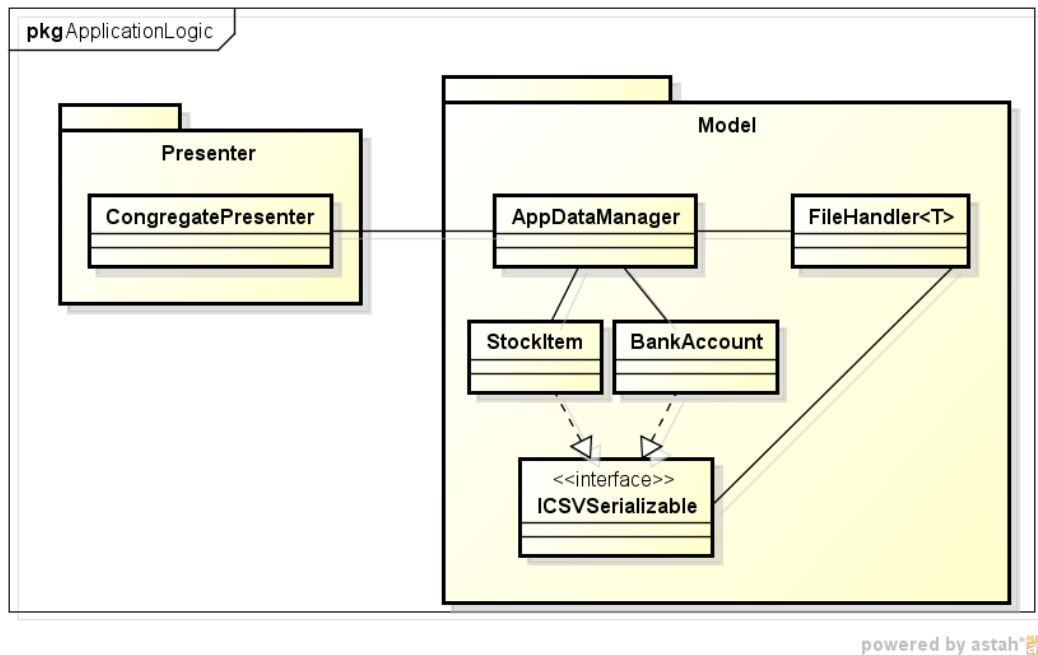


Figure 2: Abstract overview of project application logic

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*.

## USER GUIDE

---

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

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

### 4.1 MANAGE STOCK ITEMS AND BANK ACCOUNTS

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

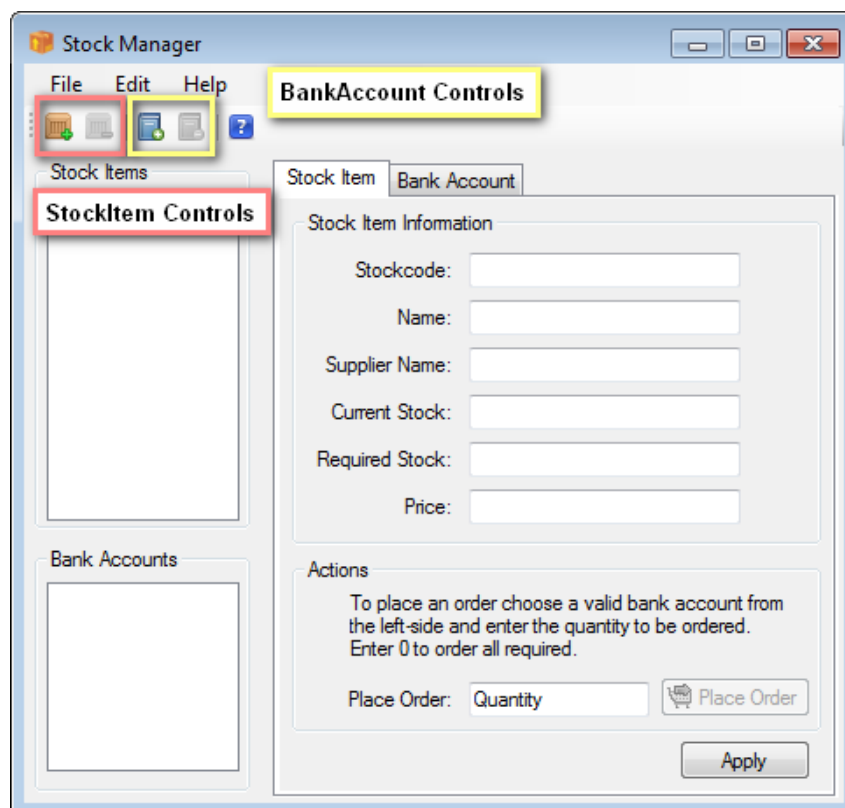


Figure 3: Main Window

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



Figure 4: Add a stock item.

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

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



Figure 5: Add a bank account.

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

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

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

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

Figure 6: Depositing and withdrawing money

## 4.2 PLACING AN ORDER

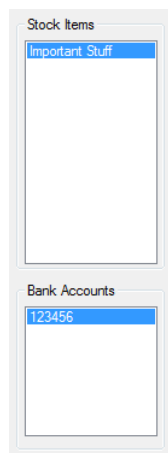


Figure 7: Selection of items.

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

Then a value 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.

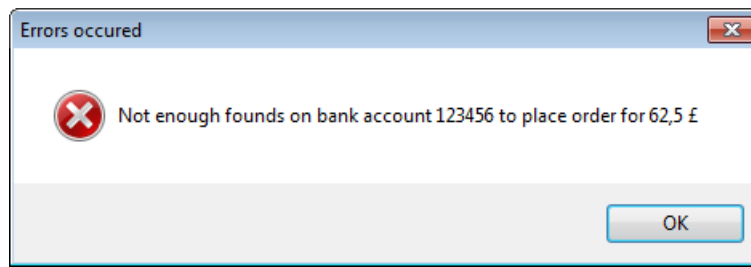


Figure 8: Placing an order without the needed funds

### 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  $\Rightarrow$  Save (Open)  $\Rightarrow$  Save (Open) bank accounts / Save (Open) stock items.

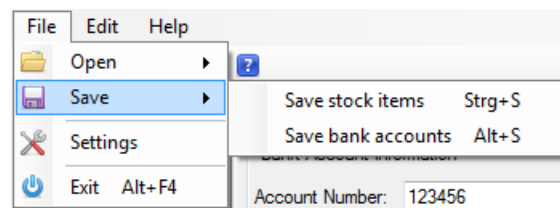


Figure 9: Saving via file menu

#### 4.3.2 Menu-bar icon

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

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

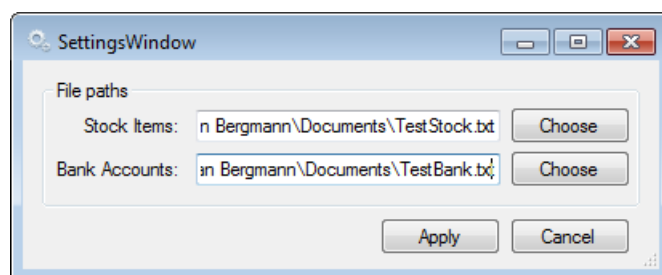


Figure 10: Settings window

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

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<sup>1</sup>.

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<sup>2</sup>.

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

```

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

```

<sup>1</sup> 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.

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



```

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

```

Listing 1: Data Binding of view and model

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

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

```

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

```

Listing 2: Example interface `IStockItemView`

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

```

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

```

```

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

```

Listing 3: Interface ICongregateView

It allows to delete items and bank accounts, 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 out in [Chapter 3](#), the AppDataManager-class works as a *facade* for the rest of the model.

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

### 5.2.1 File handler

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

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

```

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

Listing 4: Interface ICSVSerializable

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

### 5.2.2 Error handling

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

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

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

```

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

```

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

Listing 5: Validate method of StockItem

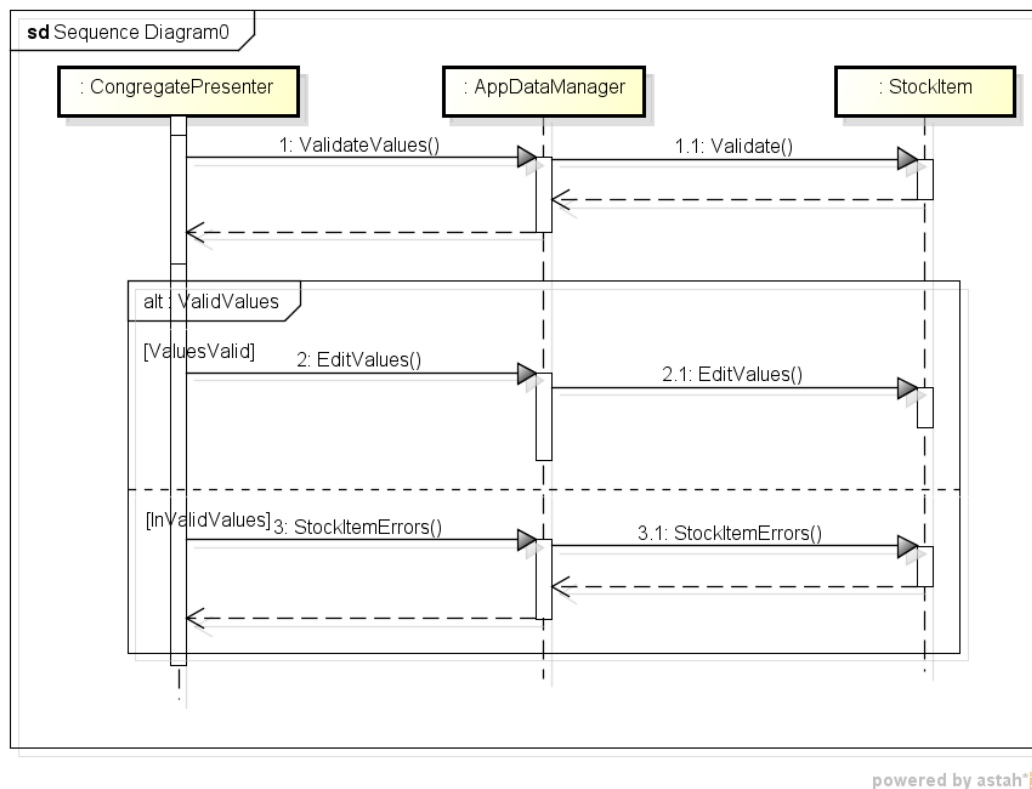


Figure 11: Sequence diagram of input validation

## TESTING

---

Testing was performed in two stages:

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

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

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

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

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

The concrete error-cases that were tested are:

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

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

Table 1: Performed tests.

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

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

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

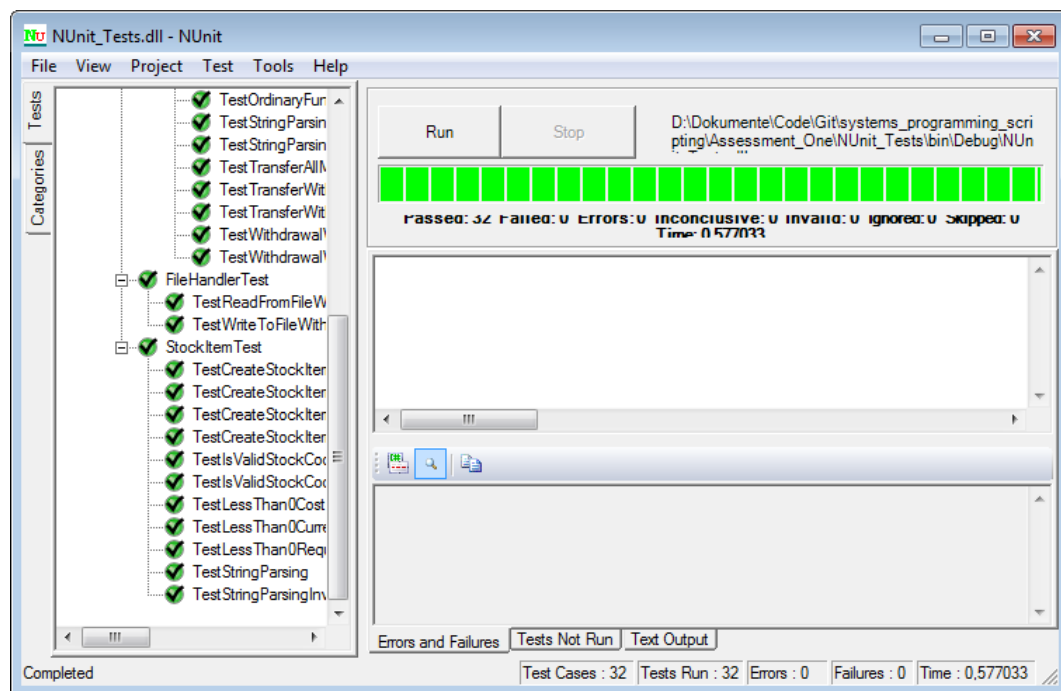


Figure 12: NUnit test case run



## CONCLUSIONS

---

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

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

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

```

41
42 #region EventHandler
43
44 private void bankAccountsListBox_SelectedIndexChanged(object sender, EventArgs
45     e)
46 {
47     if (this.bankAccountsListBox.SelectedItem != null)
48     {
49         this.tabControl1.SelectTab(BANKACCOUNTTAB);
50         this.SwitchBankAccountControls(true);
51     }
52     else
53     {
54         this.SwitchBankAccountControls(false);
55     }
56 }
57
58 private void stockItemsListBox_SelectedValueChanged(object sender, EventArgs e
59     )
60 {
61     if (this.stockItemsListBox.SelectedItem != null)
62     {
63         this.tabControl1.SelectTab(STOCKITEMTAB);
64         this.SwitchStockItemControls(true);
65     }
66     else
67     {
68         this.SwitchStockItemControls(false);
69     }
70 }
71
72 private void addStockItemToolStripMenuItem_Click(object sender, EventArgs e)
73 {
74     this._Presenter.CreateNewStockItem();
75 }
76
77 private void deleteStockItemToolStripMenuItem_Click(object sender, EventArgs e
78     )
79 {
80     _Presenter.DeleteStockItem();
81 }
82
83 private void addBankAccountToolStripMenuItem_Click(object sender, EventArgs e)
84 {
85     this._Presenter.CreateNewBankAccount();
86 }
87
88 private void deleteBankAccountToolStripMenuItem_Click(object sender, EventArgs
89     e)
90 {
91     this._Presenter.DeleteBankAccount();
92 }

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

528         nameTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Surname", false
529         , DataSourceUpdateMode.Never);
530         balanceTextBox.DataBindings.Add("Text", _Model.BankAccounts, "Balance",
531         false, DataSourceUpdateMode.Never);
532     }
533
534     private void quantityTextBox_TextChanged(object sender, EventArgs e)
535     {
536         String newText = quantityTextBox.Text;
537         int parseInt = 0;
538         if (int.TryParse(newText, out parseInt))
539         {
540             placeOrderButton.Enabled = true;
541         }
542         else
543         {
544             placeOrderButton.Enabled = false;
545         }
546     }
547
548     #endregion
549 }

```

Listing 6: MainWindow.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using System.Collections.Specialized;
10 using System.Configuration;
11 using Assessment_One.Properties;
12
13 namespace Assessment_One
14 {
15     public partial class SettingsWindow : Form
16     {
17         public SettingsWindow()
18         {
19             InitializeComponent();
20             SetUpTextBoxes();
21         }
22
23         private void SetUpTextBoxes()
24         {
25             Settings settings = Settings.Default;

```

```

26         stockItemsFilePathTextBox.Text = settings.StockItemFilePath;
27         bankAccountFilePathTextBox.Text = settings.BankAccountFilePath;
28     }
29
30     private void chooseStockItemsFilePath_Click(object sender, EventArgs e)
31     {
32         this.openFileDialog.ShowDialog();
33         OpenFileDialog file = this.openFileDialog;
34         this.stockItemsFilePathTextBox.Text = file.FileName;
35     }
36
37     private void chooseBankAccountsFilePath_Click(object sender, EventArgs e)
38     {
39         this.openFileDialog.ShowDialog();
40         OpenFileDialog file = this.openFileDialog;
41         this.bankAccountFilePathTextBox.Text = file.FileName;
42     }
43
44     private void cancelButton_Click(object sender, EventArgs e)
45     {
46         this.Dispose();
47     }
48
49     private void applyButton_Click(object sender, EventArgs e)
50     {
51         String stockItemsFilePath = stockItemsFilePathTextBox.Text;
52         String bankAccountsFilePath = bankAccountFilePathTextBox.Text;
53         this.SaveApplicationSettings(stockItemsFilePath, bankAccountsFilePath);
54         this.Dispose();
55     }
56
57     private void SaveApplicationSettings(string stockItemsFilePath, string
58         bankAccountsFilePath)
59     {
60         Settings settings = Settings.Default;
61         settings.StockItemFilePath = stockItemsFilePath;
62         settings.BankAccountFilePath = bankAccountsFilePath;
63         settings.Save();
64     }
65 }

```

Listing 7: Settings.cs

## A.2 APPLICATION LOGIC

### A.2.1 *Interfaces Package*

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Interfaces
7  {
8      /// <summary>
9      /// Utilized by the presenter to get the necessary values from a view.
10     /// </summary>
11     public interface IBankAccountView
12     {
13         int AccountNumber { get; }
14         String Surname { get; }
15         double Balance { get; }
16     }
17 }

```

Listing 8: IBankAccountView.cs

```

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

```



Listing 9: ICongregateView.cs

```

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

```

Listing 10: ICSVSerializable.cs

```

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

```

Listing 11: IStockItemView.cs

```

1  using System;
2  using System.ComponentModel;
3  using ApplicationLogic.Model;
4
5  namespace ApplicationLogic.Interfaces
6  {

```

```

7      /// <summary>
8      /// Utilized by the View to set up the data binding to the lists in the model
9      /// </summary>
10     public interface IViewModel
11     {
12         BindingList<StockItem> StockItems { get; }
13         BindingList<BankAccount> BankAccounts { get; }
14     }
15 }

```

Listing 12: IViewModel.cs

### A.2.2 Model Package

```

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

```

```

36
37     /// <summary>
38     /// Gets or sets the stock item handler.
39     /// </summary>
40     /// <value>The stock item handler.</value>
41     public FileHandler<StockItem> StockItemHandler { get; private set; }
42     /// <summary>
43     /// Gets or sets the bank account handler.
44     /// </summary>
45     /// <value>The bank account handler.</value>
46     public FileHandler<BankAccount> BankAccountHandler { get; private set; }
47
48     public AppDataManager()
49     {
50         this.StockItems = new BindingList<StockItem>();
51         this.BankAccounts = new BindingList<BankAccount>();
52         this.StockItemHandler = new FileHandler<StockItem>();
53         this.BankAccountHandler = new FileHandler<BankAccount>();
54     }
55
56     /// <summary>
57     /// Creates a new StockItem and initializes it with dummy values.
58     /// Adds the item to the StockItem collection.
59     /// </summary>
60     public void CreateNewStockItem()
61     {
62         StockItem si = new StockItem("oooo", "Dummy Item", "None", 0.0, 0, 0);
63         this.StockItems.Add(si);
64     }
65
66     /// <summary>
67     /// Deletes a StockItem from the StockItem collection.
68     /// Throws an ArgumentException if the item can not be found.
69     /// </summary>
70     public void DeleteStockItem(StockItem si)
71     {
72         if (this.StockItems.Contains(si))
73         {
74             this.StockItems.Remove(si);
75         }
76         else
77         {
78             throw new ArgumentException("Item to delete not present.");
79         }
80     }
81
82     /// <summary>
83     /// Creates a new BankAccount and initializes it with dummy values.
84     /// Adds the item to the BankAccount collection.
85     /// </summary>
86     public void CreateNewBankAccount()

```

```

87     {
88         BankAccount ba = new BankAccount(0, "Dummy Account", 0.0);
89         this.BankAccounts.Add(ba);
90     }
91
92     /// <summary>
93     /// Deletes a BankAccount from the BankAccount collection.
94     /// Throws an ArgumentException if the account can not be found.
95     /// </summary>
96     public void DeleteBankAccount(BankAccount ba)
97     {
98         if (this.BankAccounts.Contains(ba))
99         {
100
101             this.BankAccounts.Remove(ba);
102         }
103         else
104         {
105             throw new ArgumentException("Item to delete not present.");
106         }
107     }
108
109     /// <summary>
110     /// Attempts to edit specified StockItem in the StockItem collection.
111     /// Throws an ArgumentException if the item can not be found.
112     /// </summary>
113     /// <param name="si">StockItem to be edited.</param>
114     /// <param name="stockCode">New StockCode</param>
115     /// <param name="supplier">New SupplierName</param>
116     /// <param name="name">New Name</param>
117     /// <param name="currentStock">New CurrentStock</param>
118     /// <param name="reqStock">New RequiredStock</param>
119     /// <param name="price">New Price</param>
120     internal void EditStockItem(StockItem si, string stockCode, string supplier,
121                                string name, int currentStock, int reqStock, double price)
122     {
123         if (si != null)
124         {
125             si.EditStockItem(stockCode, name, supplier, price, reqStock,
126                             currentStock);
127         }
128         else
129         {
130             throw new ArgumentNullException("Stock item to edit not present.");
131         }
132     }
133
134     /// <summary>
135     /// Attempts to edit specified BankAccount in the BankAccount collection.
136     /// Throws an ArgumentException if the account can not be found.
137     /// </summary>

```

```

136    /// <param name="ba">BankAccount to be edited</param>
137    /// <param name="surname">New surname</param>
138    /// <param name="accountNumber">New accountnumber</param>
139    internal void EditBankAccount(BankAccount ba, string surname, int
        accountNumber)
140    {
141        if (ba != null)
142        {
143            ba.EditBankAccount(surname, accountNumber);
144        }
145        else
146        {
147            throw new ArgumentNullException("Bank account to edit not present.");
148        }
149    }
150
151
152    /// <summary>
153    /// Validates the changes that may be made to a stock item.
154    /// </summary>
155    /// <param name="accountNumber">New account number</param>
156    /// <param name="surname">New surname</param>
157    /// <returns>True if new values are valid. False otherwise.</returns>
158    internal bool ValidateStockItem(string stockCode, string name, string supplier
        , 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 occurred in the last validation of stock item data.
166    /// </summary>
167    /// <returns>The errors that occurred.</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 occurred in the last validation of bank account data
192     .
193     /// </summary>
194     /// <returns>The errors that occurred.</returns>
195     internal ErrorMessageCollection BankAccountErrors()
196     {
197         return BankAccount.ErrorMessages;
198     }
199
200     /// <summary>
201     /// Validates the changes that may be made to a bank account.
202     /// </summary>
203     /// <param name="accountNumber">New account number</param>
204     /// <param name="surname">New surname</param>
205     /// <returns>True if new values are valid. False otherwise.</returns>
206     internal bool ValidateBankAccount(int accountNumber, string surname)
207     {
208         bool areValidValues = BankAccount.Validate(accountNumber, surname);
209         return areValidValues;
210     }
211
212     /// <summary>
213     /// Attempts to order an item. If no quantity was provided the required
214     /// quantity will be ordered.
215     /// </summary>
216     /// <param name="indexStockItem">Index of the stock item in the stock items
217     /// list.</param>
218     /// <param name="indexBankAccount">Index of the bank account in the bank
219     /// account list.</param>
220     /// <param name="quantity">The quantity to be ordered. 0 orders the required
221     /// quantity.</param>
222     public void OrderItem(StockItem si, BankAccount ba, int quantity)
223     {
224         if (ba != null && si != null)
225         {
226             bool buyExcessStock = false;
227             if (quantity == 0)
228             {
229                 quantity = si.RequiredStock;
230             }
231             else
232             {
233                 /*

```

```

230         * It is possible to order more than the needed quantity.
231         */
232         buyExcessStock = true;
233     }
234     double priceOfOrder = quantity * si.UnitCost;
235     if (priceOfOrder <= ba.Balance)
236     {
237         ba.Transfer(1, priceOfOrder);
238         si.CurrentStock += quantity;
239         /*
240          * Allow the user to buy more than needed.
241          */
242         if (buyExcessStock)
243         {
244             if (quantity > si.RequiredStock)
245             {
246                 si.RequiredStock = 0;
247             }
248         }
249     }
250     else
251     {
252         throw new NotEnoughFundsException(String.Format("Not enough funds
                on bank account {0} to place an order for {1} £", ba.
                AccountNumber, priceOfOrder));
253     }
254 }
255 else
256 {
257     throw new ArgumentNullException("Stock item or bank account provided
        do not exist.");
258 }
259 }
260
261 /// <summary>
262 /// Attempts to deposit the requested amount from the specified bank account.
263 /// </summary>
264 /// <param name="indexBankAccount"></param>
265 /// <param name="amount"></param>
266 internal void Deposit(BankAccount ba, double amount)
267 {
268     if (ba != null)
269     {
270         ba.Deposit(amount);
271     }
272     else
273     {
274         throw new ArgumentNullException("Provided bank account does not exist.
            ");
275     }
276 }

```

```

277
278     /// <summary>
279     /// Attempts to withdraw the specified amount from the specified bank account.
280     /// </summary>
281     /// <param name="indexBankAccount"></param>
282     /// <param name="amount"></param>
283     internal void Withdraw(BankAccount ba, double amount)
284     {
285         if (ba != null)
286         {
287             ba.Withdraw(amount);
288         }
289         else
290         {
291             throw new ArgumentNullException("Bank account provided does not exist.
292             ");
293         }
294     }
295
296     /// <summary>
297     /// Will reload the StockItem collection from the specified file.
298     /// Will overwrite the currently existing StockItem collection.
299     /// </summary>
300     /// <param name="filePath"></param>
301     internal void LoadStockItemsFromFile(string filePath)
302     {
303         this.StockItemHandler.ReadFilePath = filePath;
304         IList<StockItem> stockItems = StockItemHandler.LoadFromFile(new StockItem
305             ());
306         this.StockItems.Clear();
307         foreach (StockItem item in stockItems)
308         {
309             this.StockItems.Add(item);
310         }
311     }
312
313     /// <summary>
314     /// Will reload the BankAccount collection from the specified file.
315     /// Will overwrite the currently existing BankAccount collection.
316     /// </summary>
317     /// <param name="filePath">The path to the file</param>
318     internal void LoadBankAccountsFromFile(string filePath)
319     {
320         this.BankAccountHandler.ReadFilePath = filePath;
321         IList<BankAccount> bankAccounts = BankAccountHandler.LoadFromFile(new
322             BankAccount());
323         this.BankAccounts.Clear();
324         foreach (BankAccount item in bankAccounts)
325         {
326             this.BankAccounts.Add(item);
327         }
328     }

```



```

325     }
326
327     /// <summary>
328     /// Will save the current StockItem collection to the specified file.
329     /// </summary>
330     /// <param name="filePath">The path to the file</param>
331     internal void SaveStockItemsToFile(string filePath)
332     {
333         this.StockItemHandler.WriteFilePath = filePath;
334         this.StockItemHandler.SaveToFile(this.StockItems);
335     }
336
337     /// <summary>
338     /// Will save the current BankAccount collection to the specified file.
339     /// </summary>
340     /// <param name="filePath">The path to the file</param>
341     internal void SaveBankAccountsToFile(string filePath)
342     {
343         this.BankAccountHandler.WriteFilePath = filePath;
344         this.BankAccountHandler.SaveToFile(this.BankAccounts);
345     }
346
347     /// <summary>
348     /// Will save the current BankAccount collection to the file stored in the
349     /// FileHandler.
350     /// Throws NoFilePathSetException if no path is set.
351     /// </summary>
352     internal void SaveBankAccountsToFile()
353     {
354         this.BankAccountHandler.SaveToFile(this.BankAccounts);
355     }
356
357     /// <summary>
358     /// Will save the current BankAccount collection to the file stored in the
359     /// FileHandler.
360     /// Throws NoFilePathSetException if no path is set.
361     /// </summary>
362     internal void SaveStockItemsToFile()
363     {
364         this.StockItemHandler.SaveToFile(this.StockItems);
365     }
366 }

```

Listing 13: AppDataManager.cs

```

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

```

```

6 using ApplicationLogic.Interfaces;
7
8 namespace ApplicationLogic.Model
9 {
10     /// <summary>
11     /// Pseudo bank account to allow the placement of an order.
12     /// </summary>
13     public class BankAccount : INotifyPropertyChanged, ICSVSerializable<
14         BankAccount>
15     {
16         private int _AccountNumber;
17         /// <summary>
18         /// Gets or sets the account number.
19         /// </summary>
20         /// <value>The account number.</value>
21         public virtual int AccountNumber {
22             get { return _AccountNumber; }
23             set {
24                 _AccountNumber = value;
25                 this.NotifyPropertyChanged ("AccountNumber");
26             }
27         }
28
29         private String _Surname;
30         /// <summary>
31         /// Gets or sets the surname.
32         /// </summary>
33         /// <value>The surname.</value>
34         public virtual String Surname {
35             get { return _Surname; }
36             set {
37                 _Surname = value;
38                 this.NotifyPropertyChanged ("Surname");
39             }
40         }
41
42         private double _Balance;
43         /// <summary>
44         /// Gets or sets the balance.
45         /// </summary>
46         /// <value>The balance.</value>
47         public virtual double Balance {
48             get { return _Balance; }
49             private set {
50                 if (value < 0.0) {
51                     throw new ArgumentException ("This class does
52                     not allow a balance smaller than 0.");
53                 } else {
54                     _Balance = value;
55                     this.NotifyPropertyChanged ("Balance");
56                 }
57             }
58         }
59     }
60 }

```

```

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 NotEnoughFundsException ("Not enough
                founds on bank account to withdraw.");
93         }
94     } else {
95         throw new ArgumentException("To deposit money please use the
                appropriate function.");
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
103     0.</param>
104     public void Deposit (double amount)
105     {
106         if (amount <= 0.0) {
107             throw new ArgumentException ("To withdraw money please
108             use the appropriate function.");
109         } else {
110             this.Balance += amount;
111         }
112     }
113     /// <summary>
114     /// Amount to be transfered to another account.
115     /// NOTE: This method is a fake to simulate "real" banking. The money
116     /// will not be transfered to any account.
117     /// </summary>
118     /// <param name="amount">Amount to be transfered - must be greater
119     /// than 0.</param>
120     /// <param name="accountNumber">Account number to transfer the money
121     /// to.</param>
122     public void Transfer (int accountNumber, double amount)
123     {
124         if (amount >= 0.0) {
125             if (this.Balance >= amount) {
126                 this.Balance -= amount;
127                 // TODO: In reality: fancy logic to transfer
128                 money.
129             } else {
130                 throw new NotEnoughFundsException ("There are
131                 not enough funds present to fulfill the
132                 required action.");
133             }
134         } else {
135             throw new ArgumentException ("It is not possible to
136             transfer funds from another account to yours.");
137         }
138     }
139     /// <summary>
140     /// Validates a set of possible changes to a BankAccount.
141     /// </summary>
142     /// <param name="accountNumber">AccountNumber to be verified</param>
143     /// <param name="surname">Surname to be verified</param>
144     /// <returns>True if the values would be valid, false otherwise.</
145     returns>
146     static internal bool Validate (int accountNumber, String surname)
147     {
148         if (String.IsNullOrEmpty (surname)) {
149             ErrorMessages.Add (new ErrorMessage ("Need the name of
150             the account owner."));
151         }
152     }

```

```

142     }
143     if (accountNumber <= 0) {
144         ErrorMessage.Add (new ErrorMessage ("Need a valid
145             account number: greater o."));
146     }
147     return ErrorMessage.Count == 0;
148 }
149
150 public event PropertyChangedEventHandler PropertyChanged;
151
152 private void NotifyPropertyChanged (String info)
153 {
154     if (PropertyChanged != null) {
155         PropertyChanged (this, new PropertyChangedEventArgs (
156             info));
157     }
158 }
159
160 internal void EditBankAccount (string surname, int accountNumber)
161 {
162     this.Surname = surname;
163     this.AccountNumber = accountNumber;
164 }
165
166 /// <summary>
167 /// Returns the current BankAccount object as a CSV-String.
168 /// </summary>
169 /// <returns>Representation of the current object as CSV-String.</
170     returns>
171 public string CsvRepresentation ()
172 {
173     return String.Format ("{0},{1},{2}", this.AccountNumber, this.
174         Surname, this.Balance);
175 }
176
177 /// <summary>
178 /// Attempts to create a BankAccount object from a string.
179 /// </summary>
180 /// <param name="stringRepresentation">The String to be parsed to bank
181     account.</param>
182 /// <returns>BankAccount object.</returns>
183 public BankAccount ParseFromString (string stringRepresentation)
184 {
185     string[] split = stringRepresentation.Split (',');
186     String accountNumber = split[0];
187     String surname = split[1];
188     String balance = split[2];
189     int accNumber = 0;
190     double bal = 0;
191     if (!String.IsNullOrEmpty (accountNumber)) {
192         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  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace ApplicationLogic.Model
7  {
8      public class ErrorMessage
9      {
10         public String Message { get; set; }
11         public String Source { get; set; }
12
13         public ErrorMessage(string p)
14         {
15             this.Message = p;
16         }
17
18         public override string ToString()
19         {
20             return Message;
21         }
22     }
23 }

```

Listing 15: ErrorMessage.cs

```

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

```

```

14     {
15         StringBuilder sb = new StringBuilder();
16
17         foreach (ErrorMessage item in this)
18         {
19             if (sb.Length > 0)
20             {
21                 sb.Append(Environment.NewLine);
22             }
23
24             sb.Append(item.ToString());
25         }
26
27         return sb.ToString();
28     }
29 }
30 }

```

Listing 16: ErrorMessageCollection.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using ApplicationLogic.Interfaces;
5  using System.IO;
6  using ApplicationLogic.Model;
7
8  namespace ApplicationLogic
9  {
10     /// <summary>
11     /// Handles reading and writing from files.
12     /// </summary>
13     public class FileHandler<T> where T : ICSVSerializable<T>
14     {
15
16         public String ReadFilePath;
17         public String WriteFilePath;
18
19         public FileHandler()
20         {
21         }
22
23         public FileHandler(String readFilePath, String writeFilePath)
24         {
25             this.ReadFilePath = readFilePath;
26             this.WriteFilePath = writeFilePath;
27         }
28
29         /// <summary>
30         /// Attempts to write the specified collection to the file specified in the
31         /// Handler.

```

```

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

```



```

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

```

Listing 18: NoFilePathSetException.cs

```

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

```

Listing 19: NotEnoughFundsException.cs

```

1  i»using System;

```

```

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

```

```

51         private set {
52             if (value < 0.0) {
53                 throw new ArgumentException ("Price can not be
54                     lower than 0.");
55             } else {
56                 _UnitCost = value;
57                 this.NotifyPropertyChanged ("UnitCost");
58             }
59         }
60     }
61
62     private int _RequiredStock;
63     public virtual int RequiredStock {
64         get { return _RequiredStock; }
65         set {
66             if (value < 0) {
67                 throw new ArgumentException ("Can not require
68                     less than 0 items.");
69             } else {
70                 _RequiredStock = value;
71                 this.NotifyPropertyChanged ("RequiredStock");
72             }
73         }
74     }
75
76     private int _CurrentStock;
77     public virtual int CurrentStock {
78         get { return _CurrentStock; }
79         set {
80             if (value < 0) {
81                 throw new ArgumentException ("Current stock
82                     can not be less than 0 items.");
83             } else {
84                 _CurrentStock = value;
85                 this.NotifyPropertyChanged ("CurrentStock");
86             }
87         }
88     }
89
90     public static ErrorMessageCollection ErrorMessages = new
91         ErrorMessageCollection ();
92
93     public StockItem ()
94     {
95     }
96
97     public StockItem (String stockCode, String name, String supplierName,
98         double unitCost, int required, int currentStock)

```

```

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
111    /// schema.</param>
112    /// <returns>True if the string conforms to the schema, false
113    /// otherwise.</returns>
114    public static bool IsValidStockCode (string value)
115    {
116        if (String.IsNullOrEmpty (value)) {
117            throw new ArgumentNullException ("Provided stockcode
118            was null or empty.");
119        } else {
120            // TODO: Do not use magic numbers in code
121            Regex regexp = new Regex (REGEX);
122            return regexp.IsMatch (value);
123        }
124    }
125
126    public void EditStockItem (String stockCode, String name, String
127    supplierName, double unitCost, int required, int currentStock)
128    {
129        this.StockCode = stockCode;
130        this.Name = name;
131        this.SupplierName = supplierName;
132        this.UnitCost = unitCost;
133        this.RequiredStock = required;
134        this.CurrentStock = currentStock;
135    }
136
137    /// <summary>
138    /// Validates a set of possible changes to a StockItem.
139    /// </summary>
140    /// <param name="stockCode">StockCodew to be verified</param>
141    /// <param name="name">Name to be verified</param>
142    /// <param name="supplierName">SupplierName to be verified</param>
143    /// <param name="unitCost">UnitCost to be verified</param>
144    /// <param name="required">RequiredStock to be verified</param>
145    /// <param name="currentStock">CurrentStock to be verified</param>

```

```

143     /// <returns>True if the values would be valid, false otherwise.</
144     returns>
145     public static bool Validate (String stockCode, String name, String
146     supplierName, double unitCost, int required, int currentStock)
147     {
148         if (String.IsNullOrEmpty (stockCode) || !IsValidStockCode (
149             stockCode)) {
150             ErrorMessage.Add (new ErrorMessage ("Need a stockcode
151                 that adheres to the stockcode format: 4 numbers."
152                 ));
153         }
154         if (String.IsNullOrEmpty (name)) {
155             ErrorMessage.Add (new ErrorMessage ("Need an item
156                 name."));
157         }
158         if (String.IsNullOrEmpty (supplierName)) {
159             ErrorMessage.Add (new ErrorMessage ("Need a supplier
160                 name."));
161         }
162         if (unitCost < 0.0) {
163             ErrorMessage.Add (new ErrorMessage ("Unit costs must
164                 be greater or equal 0."));
165         }
166         if (required < 0) {
167             ErrorMessage.Add (new ErrorMessage ("Required stock
168                 must be greater or equal 0."));
169         }
170         if (currentStock < 0) {
171             ErrorMessage.Add (new ErrorMessage ("Current stock
172                 must be greater or equal 0."));
173         }
174         return ErrorMessage.Count == 0;
175     }
176
177     public event PropertyChangedEventHandler PropertyChanged;
178
179     private void NotifyPropertyChanged (String info)
180     {
181         if (PropertyChanged != null) {
182             PropertyChanged (this, new PropertyChangedEventArgs (
183                 info));
184         }
185     }
186
187     /// <summary>
188     /// Returns the current StockItem object as a CSV-String.
189     /// </summary>
190     /// <returns>Representation of the current object as CSV-String.</
191     returns>
192     public String CsvRepresentation ()
193     {

```

```

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  i»using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Linq;
5  using System.Text;
6  using ApplicationLogic.Interfaces;
7  using ApplicationLogic.Model;

```

```

8
9 namespace ApplicationLogic.Presenter
10 {
11     /// <summary>
12     /// Presenter for the MainWindow: handles events and GUI-related part of the
13     /// application logic.
14     /// </summary>
15     public class CongregatePresenter
16     {
17         public ICongregateView _View;
18         public IStockItemView _StockItemView;
19         public IBankAccountView _BankAccountView;
20         public AppDataManager _Model;
21
22         /// <summary>
23         /// Initializes a new instance of the <see cref="CongregatePresenter"/> class.
24         /// </summary>
25         /// <param name="view">The view.</param>
26         /// <param name="stockItemView">The stock item view.</param>
27         /// <param name="bankAccountView">The bank account view.</param>
28         /// <param name="model">The model.</param>
29         public CongregatePresenter(ICongregateView view, IStockItemView stockItemView,
30             IBankAccountView bankAccountView, IViewModel model)
31         {
32             this._View = view;
33             this._StockItemView = stockItemView;
34             this._BankAccountView = bankAccountView;
35             this._Model = model as AppDataManager;
36         }
37
38         /// <summary>
39         /// <see cref="ApplicationLogic.Model.AppDataManagerClass"/>
40         /// </summary>
41         public void CreateNewStockItem()
42         {
43             this._Model.CreateNewStockItem();
44         }
45
46         /// <summary>
47         /// Deletes the stock item.
48         /// </summary>
49         public void DeleteStockItem()
50         {
51             if (this._View.ConfirmDelete())
52             {
53                 StockItem si = this._View.StockItem;
54                 this._Model.DeleteStockItem(si);
55             }
56         }

```

```

57     /// <summary>
58     /// Creates the new bank account.
59     /// </summary>
60     public void CreateNewBankAccount()
61     {
62         this._Model.CreateNewBankAccount();
63     }
64
65     /// <summary>
66     /// Deletes the bank account.
67     /// </summary>
68     public void DeleteBankAccount()
69     {
70         if (this._View.ConfirmDelete())
71         {
72             BankAccount ba = this._View.BankAccount;
73             this._Model.DeleteBankAccount(ba);
74         }
75     }
76
77     /// <summary>
78     /// Edits the stock item.
79     /// </summary>
80     public void EditStockItem()
81     {
82         try
83         {
84             StockItem si = this._View.StockItem;
85             String stockCode = this._StockItemView.StockCode;
86             String supplier = this._StockItemView.SupplierName;
87             String name = this._StockItemView.ItemName;
88             int currentStock = this._StockItemView.CurrentStock;
89             int reqStock = this._StockItemView.RequiredStock;
90             double price = this._StockItemView.UnitCost;
91             bool areValuesValid = this._Model.ValidateStockItem(stockCode, name,
92                 supplier, price, reqStock, currentStock);
93             if (areValuesValid)
94             {
95                 this._Model.EditStockItem(si, stockCode, supplier, name,
96                     currentStock, reqStock, price);
97             }
98             else
99             {
100                 this._View.DisplayValidationErrors(this._Model.StockItemErrors());
101                 this._Model.ClearStockItemErrors();
102             }
103         }
104         catch (FormatException e)
105         {
106             DisplayError(e);
107         }
108     }

```



```

106     }
107
108
109     /// <summary>
110     /// Edits the bank account.
111     /// </summary>
112     public void EditBankAccount()
113     {
114         try
115         {
116             BankAccount ba = this._View.BankAccount;
117             String surname = this._BankAccountView.Surname;
118             int accountNumber = this._BankAccountView.AccountNumber;
119             bool areValuesValid = this._Model.ValidateBankAccount(accountNumber,
120                                                                     surname);
121             if (areValuesValid)
122             {
123                 this._Model.EditBankAccount(ba, surname, accountNumber);
124             }
125             else
126             {
127                 this._View.DisplayValidationErrors(this._Model.BankAccountErrors());
128                 this._Model.ClearBankAccountErrors();
129             }
130         }
131         catch (FormatException e)
132         {
133             DisplayError(e);
134         }
135     }
136
137     /// <summary>
138     /// Orders the item.
139     /// </summary>
140     public void OrderItem()
141     {
142         try
143         {
144             BankAccount ba = this._View.BankAccount;
145             StockItem si = this._View.StockItem;
146             this.EditStockItem();
147             int quantity = this._View.Quantity;
148             this._Model.OrderItem(si, ba, quantity);
149         }
150         catch (FormatException e)
151         {
152             DisplayError(e);
153         }
154         catch (NotEnoughFundsException e)

```

```

155         {
156             DisplayError(e);
157         }
158     }
159
160     /// <summary>
161     /// Deposits this instance.
162     /// </summary>
163     public void Deposit()
164     {
165         try
166         {
167             BankAccount ba = this._View.BankAccount;
168             double amount = this._View.Deposit;
169             this._Model.Deposit(ba, amount);
170         }
171         catch (ArgumentNullException e)
172         {
173             DisplayError(e);
174         }
175         catch (ArgumentException e)
176         {
177             DisplayError(e);
178         }
179         catch (FormatException e)
180         {
181             DisplayError(e);
182         }
183         catch (NotEnoughFundsException e)
184         {
185             DisplayError(e);
186         }
187     }
188
189     /// <summary>
190     /// Withdraws this instance.
191     /// </summary>
192     public void Withdraw()
193     {
194         try
195         {
196             BankAccount ba = this._View.BankAccount;
197             double amount = this._View.Withdraw;
198             this._Model.Withdraw(ba, amount);
199         }
200         catch (ArgumentNullException e)
201         {
202             DisplayError(e);
203         }
204         catch (ArgumentException e)
205         {

```

```

206         DisplayError(e);
207     }
208     catch (FormatException e)
209     {
210         DisplayError(e);
211     }
212     catch (NotEnoughFundsException e)
213     {
214         DisplayError(e);
215     }
216 }
217
218 /// <summary>
219 /// Displays the error.
220 /// </summary>
221 /// <param name="e">The e.</param>
222 private void DisplayError(Exception e)
223 {
224     ErrorMessageCollection col = new ErrorMessageCollection();
225     col.Add(new ErrorMessage(e.Message));
226     this._View.DisplayValidationErrors(col);
227 }
228
229 /// <summary>
230 /// Closes the application.
231 /// </summary>
232 public void CloseApplication()
233 {
234     if (this._View.ConfirmClose())
235         Environment.Exit(1);
236 }
237
238 /// <summary>
239 /// Loads the stock items from file.
240 /// </summary>
241 /// <param name="filePath">The file path.</param>
242 public void LoadStockItemsFromFile(String filePath)
243 {
244     this._Model.LoadStockItemsFromFile(filePath);
245 }
246
247 /// <summary>
248 /// Loads the bank accounts from file.
249 /// </summary>
250 /// <param name="filePath">The file path.</param>
251 public void LoadBankAccountsFromFile(String filePath)
252 {
253     this._Model.LoadBankAccountsFromFile(filePath);
254 }
255
256 /// <summary>

```

```

257     /// Saves the stock items to file.
258     /// </summary>
259     /// <param name="filePath">The file path.</param>
260     public void SaveStockItemsToFile(String filePath)
261     {
262         this._Model.SaveStockItemsToFile(filePath);
263     }
264
265     /// <summary>
266     /// Saves the bank accounts to file.
267     /// </summary>
268     /// <param name="filePath">The file path.</param>
269     public void SaveBankAccountsToFile(String filePath)
270     {
271         this._Model.SaveBankAccountsToFile(filePath);
272     }
273
274     /// <summary>
275     /// Sets up stock item file path.
276     /// </summary>
277     /// <param name="filePathStockItems">The file path stock items.</param>
278     public void SetUpStockItemFilePath(string filePathStockItems)
279     {
280         // TODO: Check if good.
281         this._Model.StockItemHandler.ReadFilePath = filePathStockItems;
282         this._Model.StockItemHandler.WriteFilePath = filePathStockItems;
283         this._Model.LoadStockItemsFromFile(filePathStockItems);
284     }
285
286     /// <summary>
287     /// Sets up bank accounts file path.
288     /// </summary>
289     /// <param name="filePathBankAccounts">The file path bank accounts.</param>
290     public void SetUpBankAccountsFilePath(string filePathBankAccounts)
291     {
292         // TODO: Check if good.
293         this._Model.BankAccountHandler.ReadFilePath = filePathBankAccounts;
294         this._Model.BankAccountHandler.WriteFilePath = filePathBankAccounts;
295         this._Model.LoadBankAccountsFromFile(filePathBankAccounts);
296     }
297
298     /// <summary>
299     /// Saves the bank accounts to file.
300     /// </summary>
301     public void SaveBankAccountsToFile()
302     {
303         try
304         {
305             this._Model.SaveBankAccountsToFile();
306         }
307         catch (NoFilePathSetException e)

```

```

308         {
309             DisplayError(e);
310         }
311     }
312
313     /// <summary>
314     /// Saves the stock items to file.
315     /// </summary>
316     public void SaveStockItemsToFile()
317     {
318         try
319         {
320             this._Model.SaveStockItemsToFile();
321         }
322         catch (NoFilePathSetException e)
323         {
324             DisplayError(e);
325         }
326     }
327 }
328 }

```

Listing 21: CongregatePresenter.cs

## A.3 TESTS

```

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

```

```

24     {
25         ba = new BankAccount(123, "Test", -1.0);
26     }
27
28     [Test]
29     [ExpectedException(typeof(NotEnoughFundsException))]
30     public void TestWithdrawalWithTooHighValues()
31     {
32         double amountToWithdrawTooHigh = 50.0;
33         ba.Withdraw(amountToWithdrawTooHigh);
34     }
35
36     [Test]
37     [ExpectedException(typeof(ArgumentException))]
38     public void TestWithdrawalWithTooSmallValues()
39     {
40         double amountToWithdrawTooSmall = -10.0;
41         ba.Withdraw(amountToWithdrawTooSmall);
42     }
43
44     [Test]
45     [ExpectedException(typeof(ArgumentException))]
46     public void TestDepositWithTooSmallValue()
47     {
48         double amountToDepositTooSmall = -10;
49         ba.Deposit(amountToDepositTooSmall);
50     }
51
52     [Test]
53     [ExpectedException(typeof(NotEnoughFundsException))]
54     public void TestTransferWithTooHighValue()
55     {
56         double amountToTransferTooHigh = 10;
57         ba.Transfer(123, amountToTransferTooHigh);
58     }
59
60     [Test]
61     [ExpectedException(typeof(ArgumentException))]
62     public void TestTransferWithTooSmallValue()
63     {
64         double amountToTransferTooSmall = -10;
65         ba.Transfer(123, amountToTransferTooSmall);
66     }
67
68     [Test]
69     public void TestOrdinaryFunctions()
70     {
71         double currentValue = ba.Balance;
72         double deposit = 50.0;
73         currentValue += deposit;
74         ba.Deposit(deposit);

```

```

75         Assert.AreEqual(currentValue, ba.Balance);
76
77         double withdraw = 25.0;
78         currentValue -= withdraw;
79         ba.Withdraw(withdraw);
80         Assert.AreEqual(currentValue, ba.Balance);
81
82         double transfer = 10.50;
83         currentValue -= transfer;
84         ba.Transfer(123, transfer);
85         Assert.AreEqual(currentValue, ba.Balance);
86     }
87
88     [Test]
89     public void TestDepositWithdrawSameAmount()
90     {
91         double amount = 50;
92         double value = 0;
93         ba.Deposit(amount);
94         ba.Withdraw(amount);
95         Assert.AreEqual(value, ba.Balance);
96     }
97
98     [Test]
99     public void TestTransferAllMoney()
100    {
101        double amount = 50;
102        double value = 0;
103        ba.Deposit(amount);
104        ba.Transfer(0, amount);
105        Assert.AreEqual(value, ba.Balance);
106    }
107
108    [Test]
109    public void TestStringParsing()
110    {
111        BankAccount parseAccount = new BankAccount();
112
113        String parseOne = "123456,Rambo,500.50";
114        BankAccount ba1 = parseAccount.ParseFromString(parseOne);
115
116        String parseTwo = "000000,,,";
117        BankAccount ba2 = parseAccount.ParseFromString(parseTwo);
118    }
119
120    [Test]
121    [ExpectedException(typeof(FormatException))]
122    public void TestStringParsingInvalidValues()
123    {
124        BankAccount parseAccount = new BankAccount();
125

```

```

126         String parseOne = "abcd,Rambo,50.50";
127         BankAccount bal = parseAccount.ParseFromString(parseOne);
128     }
129 }
130 }

```

Listing 22: BankAccountTest.cs

```

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

```

Listing 23: FileHandlerTest.cs

```

1  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using ApplicationLogic.Model;
6  using NUnit.Framework;
7
8  namespace NUnit_Tests
9  {
10     [TestFixture]
11     public class StockItemTest

```



```

12 {
13     private StockItem si;
14
15     [SetUp]
16     public void Setup()
17     {
18         si = new StockItem("1234", "Test", "Test", 10.0, 5, 5);
19     }
20
21     [Test]
22     [ExpectedException(typeof(ArgumentException))]
23     public void TestLessThan0Cost()
24     {
25         si = new StockItem("1234", "Test", "Test", -1.0, 5, 5);
26     }
27
28     [Test]
29     [ExpectedException(typeof(ArgumentException))]
30     public void TestLessThan0CurrentStock()
31     {
32         si = new StockItem("1234", "Test", "Test", 1.0, -1, 5);
33     }
34
35     [Test]
36     [ExpectedException(typeof(ArgumentException))]
37     public void TestLessThan0RequiredStock()
38     {
39         si = new StockItem("1234", "Test", "Test", 1.0, 5, -1);
40     }
41
42     [Test]
43     public void TestIsValidStockCode()
44     {
45         bool validSC = StockItem.IsValidStockCode("1234");
46         Assert.IsTrue(validSC);
47         bool tooLongSC = StockItem.IsValidStockCode("123456");
48         Assert.IsFalse(tooLongSC);
49         bool stringSC = StockItem.IsValidStockCode("test");
50         Assert.IsFalse(stringSC);
51     }
52
53     [Test]
54     [ExpectedException(typeof(ArgumentNullException))]
55     public void TestIsValidStockCodeRaiseException()
56     {
57         StockItem.IsValidStockCode(null);
58     }
59
60     [Test]
61     [ExpectedException(typeof(ArgumentException))]
62     public void TestCreateStockItemInvalidStockCode()

```

```

63     {
64         String invalidStockCode = "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  i»using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using NUnit.Framework;
6  using Moq;
7  using ApplicationLogic.Model;
8
9  namespace NUnit_Tests.ApplicationLogic
10 {
11     [TestFixture]
12     public class AppDataManagerTest
13     {
14         private AppDataManager _Manager;
15         private StockItem _Stock;
16         private BankAccount _Account;
17
18         [SetUp]
19         public void SetUp()
20         {
21             this._Manager = new AppDataManager();
22             this._Stock = new StockItem();
23             this._Account = new BankAccount();
24         }
25
26         [Test]
27         public void TestAddStockItem()
28         {
29             this._Manager.CreateNewStockItem();
30             Assert.AreEqual(1, this._Manager.StockItems.Count);
31         }
32
33         [Test]
34         public void TestAddBankAccount()
35         {
36             this._Manager.CreateNewBankAccount();
37             Assert.AreEqual(1, this._Manager.BankAccounts.Count);
38         }

```

```

39
40 [Test]
41 public void TestRemoveStockItem()
42 {
43     this._Manager.CreateNewStockItem();
44     Assert.AreEqual(1, this._Manager.StockItems.Count);
45     StockItem remove = this._Manager.StockItems.ElementAt(0);
46     this._Manager.DeleteStockItem(remove);
47     Assert.AreEqual(0, this._Manager.StockItems.Count);
48 }
49
50 [Test]
51 public void TestRemoveBankAccount()
52 {
53     this._Manager.CreateNewBankAccount();
54     Assert.AreEqual(1, this._Manager.BankAccounts.Count);
55     BankAccount remove = this._Manager.BankAccounts.ElementAt(0);
56     this._Manager.DeleteBankAccount(remove);
57     Assert.AreEqual(0, this._Manager.BankAccounts.Count);
58 }
59
60 [Test]
61 [ExpectedException(typeof(ArgumentException))]
62 public void TestRemoveStockItemNotPresent()
63 {
64     StockItem si = new StockItem();
65     this._Manager.DeleteStockItem(si);
66 }
67
68 [Test]
69 [ExpectedException(typeof(ArgumentException))]
70 public void TestRemoveBankAccountNotPresent()
71 {
72     BankAccount ba = new BankAccount();
73     this._Manager.DeleteBankAccount(ba);
74 }
75
76 [Test]
77 public void TestCorrectSequenceOfOrdering()
78 {
79     var mockBa = new Mock<BankAccount>();
80     var mockSi = new Mock<StockItem>();
81
82     mockBa.Setup(ba => ba.Balance).Returns(50.0);
83
84     this._Manager.BankAccounts.Add(mockBa.Object);
85     this._Manager.StockItems.Add(mockSi.Object);
86     this._Manager.OrderItem(mockSi.Object, mockBa.Object, 0);
87
88     mockBa.VerifyGet(ba => ba.Balance);
89     mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.AtMostOnce());

```

```
90     }
91
92     [Test]
93     [ExpectedException(typeof(NotEnoughFundsException))]
94     public void TestCorrectSequenceOnInvalidFunds()
95     {
96         var mockBa = new Mock<BankAccount>();
97         var mockSi = new Mock<StockItem>();
98
99         mockBa.Setup(ba => ba.Balance).Returns(0.0);
100        mockSi.Setup(si => si.UnitCost).Returns(10.0);
101
102        this._Manager.BankAccounts.Add(mockBa.Object);
103        this._Manager.StockItems.Add(mockSi.Object);
104        this._Manager.OrderItem(mockSi.Object, mockBa.Object, 1);
105
106        mockBa.VerifyGet(ba => ba.Balance);
107        mockBa.Verify(ba => ba.Transfer(0, 10.0), Times.Never());
108    }
109 }
110 }
```

Listing 25: AppDataManagerTest.cs

## BIBLIOGRAPHY

---

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