Developer handbook

Customer management application

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project No.** | **Doc. ID** | **Version** | **Date** | **Author** | **Status** |
| 2018-001 | 2000 | 2.0 | 21.05.2018 | JoAn MiVu WiLe | Valid |

Table of contents

[1 Introduction 3](#_Toc514693319)

[1.1 Purpose and scope 3](#_Toc514693320)

[1.2 Structure of document 3](#_Toc514693321)

[1.3 Related documents 3](#_Toc514693322)

[1.3.1 Project notebook 3](#_Toc514693323)

[1.3.2 User documentation 3](#_Toc514693324)

[1.4 Versioning 3](#_Toc514693325)

[2 Design 4](#_Toc514693326)

[2.1 General class diagram 4](#_Toc514693327)

[2.2 Interface description of user interface 4](#_Toc514693328)

[2.3 File structure and naming 6](#_Toc514693329)

[3 Implementation 7](#_Toc514693330)

[3.1 Project library 7](#_Toc514693331)

[3.1.1 Customer class 7](#_Toc514693332)

[3.1.2 Encrypt class 8](#_Toc514693333)

[3.2 Graphical user interface (GUI) 8](#_Toc514693334)

[3.2.1 Password window 8](#_Toc514693335)

[3.2.2 Main window 9](#_Toc514693336)

[3.2.3 Add or Edit Customer 10](#_Toc514693337)

[4 Testing 11](#_Toc514693338)

[4.1 Unit tests 11](#_Toc514693339)

[4.1.1 Unit test of project library 11](#_Toc514693340)

[4.1.2 Unit test GUI 11](#_Toc514693341)

[4.2 Integration Tests 11](#_Toc514693342)

[4.3 System Tests 11](#_Toc514693343)

[4.3.1 Partition test cases, email 12](#_Toc514693344)

[4.3.2 Partition test cases, Balance 12](#_Toc514693345)

# Introduction

## Purpose and scope

## Structure of document

The document includes information about the developing process of the project. It should basically describe the process from its beginning to the final steps. This includes how the project was designed, implemented into code and tested by the project team.

## Related documents

### Project notebook

The project notebook provides information about general topics of the project. It documents the tasks of each member, the used tools, standards and gives a short overview of the tasks.

### User documentation

The user documentation is containing the scope of delivery, how the user gets started with the software and operator information.   
Furthermore, also an outlook for the future and further releases are given.

## Versioning

The whole project was expanded into a second version.   
The basic requirements for the first version were to create a program which allows the user to add and edit customers. Furthermore, the balance of each customer should be changeable. All the stored data needed to be encrypted to allow data security. In addition, users should only get access to the program with a certain password.

The version two, which is the expanded version, should now require internationalization of the program.   
Therefore, the given requirements were set to allow the change of the user language in at least one sub window.

# Design

## General class diagram

## Interface description of user interface

Processing:

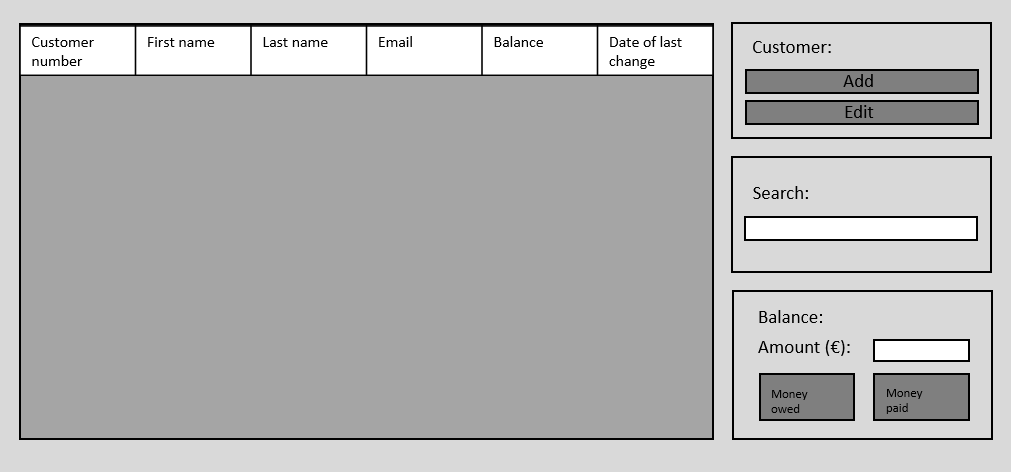
* Reads the inserted password and checks if it is correct
* Reads the inserted first name, last name and email address and checks if it is valid
* Throws and error if password is wrong or null
* Throws and error if the names or email address is wrong or null
* Checks if email address is unique
* Throws and error if email address is not unique
* Numeric up and down box for the balance allows only a numeric value, therefore no checking of the inserted value is necessary

Input:

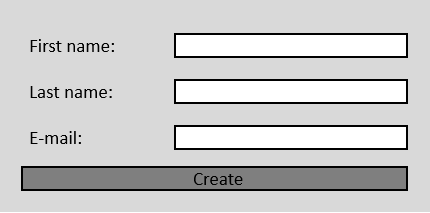
* Password string
* First name string
* Last name string
* E-mail string
* Search-box string
* Amount-Balance decimal
* Button – Log In
* Button – Cancel
* Button – Add customer
* Button – Edit customer
* Button – Create / Change customer
* Button – Money owed
* Button – Money paid

Output:

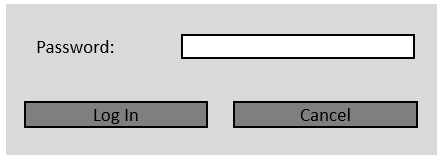
* Error if something is not valid or password incorrect
* Showing main window if password is correct
* Illustrating new added or edited customer if correct



Concept design of the main window.



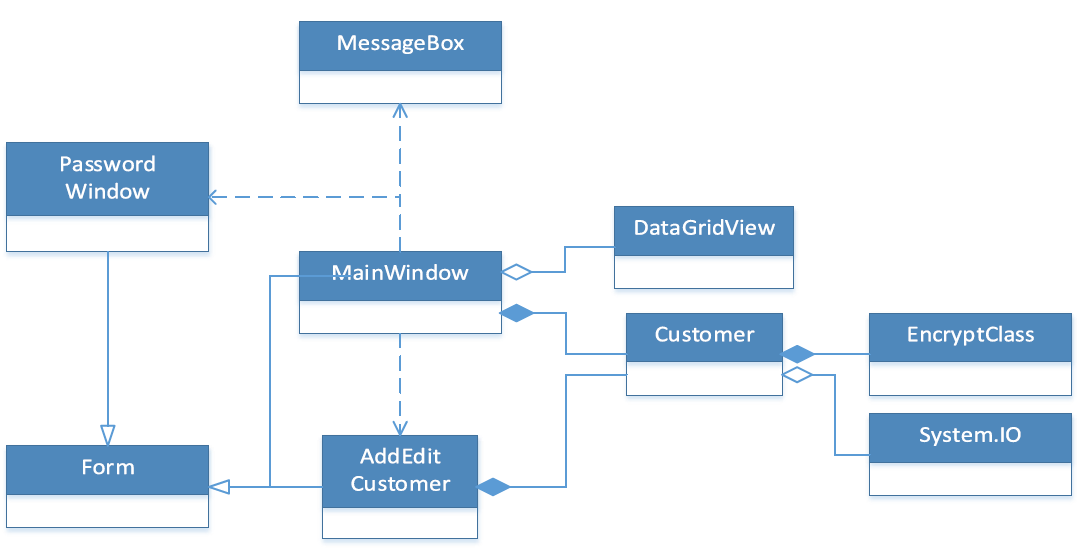
Concept design of the sub window for adding or editing a customer.



Concept design of the password window.

## File structure and naming

To get a more specific idea of how the several parts of the project work with each other a rare class diagram had been developed in a first instance.



The *password window*, *Main Application* and the *AddEditCustomer* inherit from the Form library as they are windows forms applications.   
The *PasswordWindow, MessageBox* and *AddEditCustomer* are associated with the *MainApplication*, as those three sub windows are used by it.

*Customer* is the library for the whole program and provides the several functions for handling or checking the inserted values.   
Therefore, the library needs to be part of the two windows which allow the user input of data.   
It was also required to store the inserted data and provide it to the user each time he starts the application. To enable this, the data is stored into a CSV file.   
Accordingly, the *System.IO* class needed to be part of the *Customer* library.

The *DataGridView* class was implemented in the *MainApplication* to provide an easy and professional illustration of the stored data.   
Therefore this class needs to be part of the *MainApplication*.

# Implementation

## Project library

For this project a project library was created. This library includes all functions which are not directly related to the graphical user interface. More detailed information with description of each function and variable can be find in the document *2001-ExplainationProjectLibClasses.*

### Customer class



### Encrypt class



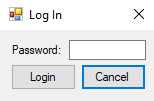
## Graphical user interface (GUI)

After the general design of the GUI and the structure of the program had been developed the application was finally implemented.  
Therefore, a windows form application in ***Microsoft Visual Studio Community,*** had been created. Furthermore, the earlier explained library was implemented into the windows forms.

The windows forms application contains all the event methods which handle the several actions triggered by the user.   
An example for this would be clicking a button of the program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Access modifier |  | Data type |  | Designation of function/variable/constants |
| Static variables | private |  | string |  | password |
| Constructors | public |  | - |  | FrmPassword() |
| Member attributes | public |  | Customer |  | ACustomer |
| Event Methods | private |  | void |  | btnLogin\_Click(object sender, EventArgs e) |
| private |  | void |  | btnCancel\_Click(object sender, EventArgs e) |

### Password window

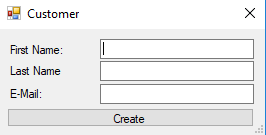
The password window allows the user only access with inserting the right password.

### Main window

The main window illustrates all the stored customers.   
It provides adding and editing, changing of the balance and searching for a specific customer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Access modifier |  | Data type | Designation of function/variable/constants |
| Member variables | private |  | List<Customer> | listCustomer |
| Constructors | public |  | - | FrmTop() |
|  |  |  |  |
| Member attributes | public |  | string | Firstname |
| public |  | string | Lastname |
| public |  | string | EmailAddress |
| public |  | int | CustomerNumber |
| public |  | double | Balance |
| public |  | DateTime | DateLastChange |
| Member methods | private |  | void | UpdateDataList() |
| private |  | void | UpdateDataList(Customer currentlySelectedCustomer) |
| Event Methods | private |  | void | btnAddCustomer\_Click(object sender, EventArgs e) |
| private |  | void | btnEditCustomer\_Click(object sender, EventArgs e) |
| private |  | void | btnMoneyPaidIn\_Click(object sender, EventArgs e) |
| private |  | void | btnMoneyPayOut\_Click(object sener, EventArgs e) |
| private |  | void | tbxEntry\_TextChanged(object sender, EventArgs e) |
| private |  | void | FrmTop\_SizeChanged(object sender, EventArgs e) |
| private |  | void | dgvListCustomer\_SelectionChanged(object sender, EventArgs e) |

### Add or Edit Customer



The add or edit window is responsible for, as the name says, adding or editing customers. The only values which can be entered while adding a customer are its first name, last name and E-Mail address.   
However, the editing option allows only the change of the last name and the E-Mail address.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Access modifier | Data type | Designation of function/variable/constants |
| Static variables | private | Customer | aCustomer |
| private | Customer | currCustomer |
| private | string | pathCSVFile |
| private | string | defLanguage |
| private | bool | customerChange |
| Constructors | public | - | FrmAddEditCust(Customer currentCustomer |
| public | - | FrmAddEditCust(Customer currentCustomer |
| Member attributes | public | Customer | ACustomer |
| Member Methods | private | void | WriteLanguageToCSV() |
| private | void | ChangeLanguage(string selectedLanguage) |
| private | void | ReadLanguageFromCSV() |
| Event Methods | private | void | btnCreateCustomer\_Click(object sender, EventArgs e) |
| private | void | cmbLanguage\_SelectedIndexChanged(object sender, EventArgs e) |

# Testing

## Unit tests

### Unit test of project library

The unit tests of the project library are done with an automatic test (Project: *UnitTestsProjectLib*). This automatic test is just a code coverage test to test the main functions. More detailed test was done in the system test.

### Unit test GUI

Unit tests regarding the GUI were done to ensure the correct functioning of buttons and text boxes.   
In a first instance cases had been tested where the use of the project library was not required. For example, checking the behaviour of the buttons. This involved characteristics like the correct opening and closing of sub windows.

Due to the parallel development of the GUI and the project library it was possible to test also the cases of inserting data into the program.   
Thus, it was tested if the application is storing the inserted value correctly to the list in the CSV file and is illustrating it in the data grid view.   
Also tests about changing the balance were possible.

## Integration Tests

Integration tests were done after development procedure as the GUI and the project library were programmed parallel.

Therefore, the several requirements like storing, checking and searching the data were tested as soon as they have been integrated into the program.   
All the cases were only tested roughly to see if the basic functioning is working. Detailed tests were done when the program was finally finished.   
Those tests are listed in detail in section *4.3 System Tests*.

## System Tests

To test the functionality of the program, partition test cases were used. Test cases were determined for the email address as well as the balance of an account. To find every possible flaw, there is a test case for every kind of undesirable sign or formatting option. At first there was only one valid test case that included every valid sign and formatting rule. The test case was later divided into several test cases for it to be interpret more easily.

There were no test cases determined for the first and last name since there are names with so many kinds of letters. Some names also have several parts.

Test case list for the balance doesn’t include a test case for a negative value. This is due the form having a button for either increasing the amount on the account or decreasing it.

### Partition test cases, email

|  |  |
| --- | --- |
| VALID | |
| **Description:** | **Test Case:** |
| a-z | oliver@fh-ooe.at |
| A-Z | OLIVER@fh-ooe.at |
| 0-9 | 0l1v3r@fh-ooe.at |
| "." | oli.ver@fh-ooe.at |
| !#$%&'\*+-/=?^\_`{|}~ | !#$%&'\*+-/=?^\_`{|}~@fh-ooe.at |
| one "@" | oliver@fh-ooe.at |
| one "." after the "@" | oliver@fh-ooe.at |
| atleast one character before "@" | o@fh-ooe.at |
| after the last "." 2-4 character | oliver@fh-ooe.atte |
| after the last "." only letters | oliver@fh-ooe.at |

|  |  |
| --- | --- |
| INVALID | |
| **Description:** | **Test Case:** |
| Null |  |
| **Before the last ".":** |  |
| no characters | .com |
| starting with "." | .oliverMarkkanen@fh-ooe.at |
| without "@" | Eemeli.niemi.ya\_hoo.com |
| more than one "@" | Yrjö@lehto@windowslive.fi |
| "." just before "@" | Aleksi.Koskinen.@windowslive.fi |
| "." just after "@" | Eemeli.niemi@.ya\_hoo.com |
| no characters before "@" | @ya\_hoo.com |
| an invalid sign ><€\"()[] | >oliverMarkkanen@fh-ooe.at |
| **After the last "."** |  |
| no characters | Aleksi.Koskinen@windowslive. |
| 1 characters | Eemeli.niemi@ya\_hoo.a |
| 5 or more characters | oliverMarkkanen@fh-ooe.apina |
| signs | Yrjö.lehto@gmail.#? |
| numbers | Aleksi.Koskinen@windowslive.92 |
| ending with "." | Yrjö.lehto@gmail.com. |

### Partition test cases, Balance

|  |  |
| --- | --- |
| VALID | |
| **Description:** | **Test Case:** |
| Positive value | 20 |

|  |  |
| --- | --- |
| INVALID | |
| **Description:** | **Test Case:** |
| Null |  |
| test with a letter | K30 |
| test with a sign | 47! |