****

**FH - Studiengang für**

**Informationstechnik und System-Management**

**Salzburg**

**ITS**

**Übungen in**

**Spezielle Softwaretechnologien**

**Protokoll**

Gegenstand der Übung gemäß Anleitung:

**Softwarekomponenten – Assemblies und Wrapper**

**Version: 1**

**Datum der Übung: 20.10.2016**

**Datum der Abgabe: 01.11.2016**

**Autoren: Christopher Wieland, Martin Wieser, Stephanie Kaschnitz**

**Unterschrift des Autors / der Autorin:**

Historie

|  |  |  |  |
| --- | --- | --- | --- |
| Änderung | Datum: | Autor: | Version: |
| Implementierung FremdWrapper | 28.10.16 | CW | 0.1 |
| Implementierung EigenWrapper | 28.10.16 | SK | 0.2 |
| Absprache mit anderer Gruppe für FremdDll | 27.10.16 | CW | 0.3 |
| Implementierung Programm | 30.10.16 | SK | 0.4 |
| Fehlerbehandlung & Bugfixes Eigendll | 11.10.16 | MW, SK | 0.5 |
| UserFunktionen hinzufügen/fixen bei unserem dll | 24.10.16 | MW | 0.6 |
| Tests | 31.10.16 | MW | 0.7 |
| Definition des Interfaces (IBankManagement) | 25.10.16 | CW,SK | 0.8 |
| Fehlerbehandlung & Fixes FremdDll | 1.11.16 | CW,MW | 0.9 |
| Tests und Abgabe | 2.11.16 | MW | 1.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[1 Aufgabenstellung 1](#_Toc466039443)

[2 Komponentenübersicht 2](#_Toc466039444)

[2.1 Interface - IBankManagement 2](#_Toc466039445)

[2.2 Eigene Komponente (Bank.dll) 3](#_Toc466039446)

[2.2.1 Requirements 3](#_Toc466039447)

[ Bank.dll 3](#_Toc466039448)

[ Eigene\_Bank\_DLL\_Assembly.dll 3](#_Toc466039449)

[ Bankanwendung.exe 3](#_Toc466039450)

[2.3 Fremde Komponente (XMLController.dll) 3](#_Toc466039451)

[2.3.1 Requirements 3](#_Toc466039452)

[3 Funktionalität der Bankanwendung 4](#_Toc466039453)

[3.1 Account Management 5](#_Toc466039454)

[3.1.1 int createSavingsAccount(int \_id); 5](#_Toc466039455)

[Erstellt ein Sparkonto mit der ID vom Customer 5](#_Toc466039456)

[3.1.2 int createCreditAccount(int \_id); 5](#_Toc466039457)

[Erstellt ein Kreditkonto mit der ID vom Customer 5](#_Toc466039458)

[3.1.3 void deleteSavingsAccount(int \_sNumber, int \_id); 5](#_Toc466039459)

[Löscht das Sparkonto vom User 5](#_Toc466039460)

[3.1.4 void deleteCreditAccount(int \_cNumber, int \_id); 5](#_Toc466039461)

[Löscht das Kreditkonto vom User 5](#_Toc466039462)

[3.1.5 void depositCreditAcc(int \_cNumber, string \_usage, double \_amount); 5](#_Toc466039463)

[Fügt dem KreditKonto Geld hinzu 5](#_Toc466039464)

[3.1.6 void depositSavingsAcc(int \_sNumber, string \_usage, double \_amount); 5](#_Toc466039465)

[Fügt dem SparKonto Geld hinzu 5](#_Toc466039466)

[3.1.7 void withdrawCreditAcc(int \_cNumber, double \_amount); 5](#_Toc466039467)

[Hebt Geld vom KreditKonto ab 5](#_Toc466039468)

[3.1.8 void transfer(int \_cNumber, int \_toAccNumber, string \_usage, double \_amount); 5](#_Toc466039469)

[Hier kann eine Überweisung mit einem Kreditkonto getätigt werden. 5](#_Toc466039470)

[3.1.9 void addSavingsAccountUser(int \_sNumber, int \_id); 5](#_Toc466039471)

[Hier kann ein Verfüger dem SparKonto hinzugefügt werden. 5](#_Toc466039472)

[3.1.10 void addCreditAccountUser(int \_cNumber, int \_id); 5](#_Toc466039473)

[Hier kann ein Verfüger dem KreditKonto hinzugefügt werden. 5](#_Toc466039474)

[3.1.11 int getBankAccountNumber(int \_id, int \_whichAccount); 5](#_Toc466039475)

[3.2 Customer-Management 6](#_Toc466039476)

[3.2.1 int createCustomer(string \_Vorname, string \_Nachname, string \_Geburtsdatum, 6](#_Toc466039477)

[string \_adresse, string \_Wohnort, string \_Telefon); 6](#_Toc466039478)

[Mit createCustomer wird ein neuer Kunde angelegt. 6](#_Toc466039479)

[3.2.2 void changeCustomer(int \_id, string \_Vorname, string \_Nachname, 6](#_Toc466039480)

[string \_address, string \_Wohnort, string \_Telefon); 6](#_Toc466039481)

[Diese Funktion ändert die Daten des übergebenen Kunden. 6](#_Toc466039482)

[3.2.3 int getCustomer(string \_firstName, string \_lastName, string \_birthDate); 6](#_Toc466039483)

[Mit getCustomer wird mit Suchparametern der Kunde aus der Datenbank gesucht und dessen ID zurückgegeben. 6](#_Toc466039484)

[3.2.4 void deleteCustomer(int \_id); 6](#_Toc466039485)

[3.3 Sonstige 6](#_Toc466039486)

[3.3.1 void createBankStatement(int \_accNumber); 6](#_Toc466039487)

[Erstellt einen Kontoauszug. 6](#_Toc466039488)

[3.3.2 void convertMoney(int Balance, string \_currency); 6](#_Toc466039489)

[Rechnet das Geld in eine andere Währung um. 6](#_Toc466039490)

[3.3.3 void showChangeOfCourse(int \_cNumber); 6](#_Toc466039491)

[4 Zusammenfassung 7](#_Toc466039492)

# Aufgabenstellung

Entwerfen Sie nun auf Basis des/der von einem anderen Team übernommen DLLs und

der eigenen DLLs ein gemeinsames C#/.NET Interface und darauf basierend .NET Assemblies, die – diesmal objektorientiert unter C# und .NET – dem eigenen DLL und

dem übernommen DLL als Wrapper/Adapter/Bridge dienen. Sie sollten die volle bzw.

geforderte Funktionalität der DLLs anbieten, ohne jedoch Details der DLLs und deren

API kennen zu müssen.

Letztlich sollten Sie mehrere .NET Assemblies implementieren, welche eine möglichst

einfache Verwendung der DLLs in C#/.NET Anwendungen und – da wir ja Komponenten

haben – den Austausch der verwendeten DLLs erlaubt. Das Ziel ist, dass Ihre zukünftige

Applikation durch den Austausch von Assemblies und der DLLs sowohl mit den eigenen

DLLs, als auch mit den DLLs des anderen Teams arbeiten kann (nicht gleichzeitig), da die

Schnittstellen beider Assembly-Gruppen in beiden Fällen ja gleich sind.

Einschränkung: Ein dynamisches Laden/Entladen der Assemblies bzw. DLLs während der

Runtime wird nicht benötigt.

Die Interfaces, Klassen, Typen,.. und das Fehlerhandling sollte so ausgelegt sein, dass die

DLLs für die Anwender völlig transparent und ohne Detailwissen über die DLLs

angebunden werden können. Aus der Nutzung der Assemblies darf in keiner Art und

Weise ersichtlich werden, dass es sich um native Dynamic Link Libraries handelt, die

lediglich ein primitives C-API anbieten.

# Komponentenübersicht

Die folgende Abbildung stellt die Komponentenübersicht mit den austauschbaren Komponenten von unserer und einer anderen Gruppe dar.

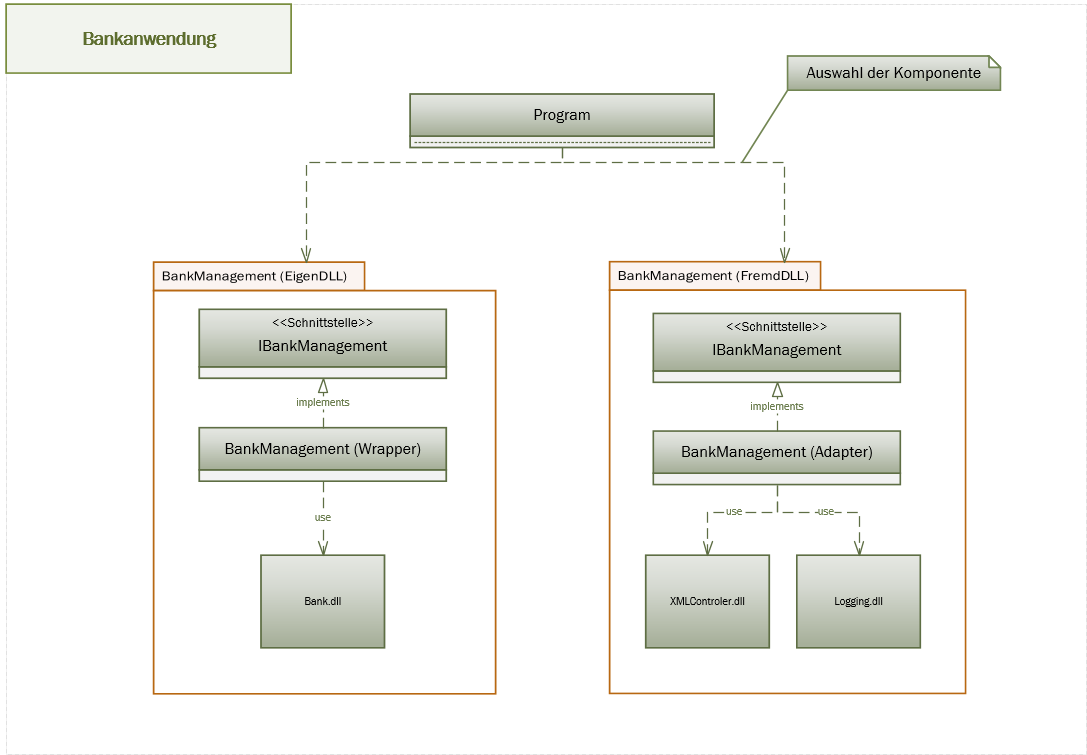


Abbildung 1: Bankanwendung – Komponentendiagramm

Bei der Bankanwendung muss durch ein/auskommentieren die Komponente gewählt werden da ein dynamisches lade nicht teil der Übung war. Beide Komponenten beinhalten dieselben Methoden. Bei unserer eigenen Komponente dient die Assembly als Wrapper während bei der DLL der anderen Gruppe vieles Adaptiert wurde.

## Interface - IBankManagement

Beide Assembly beinhalten die Klasse BankManagement die das Interface IBankManagement implementieren und somit können diese ausgetauscht werden.

## Eigene Komponente (Bank.dll)

Hier sind die Imports vom der Bank.dll Komponenten enthalten und werden adaptiert, d.h. die Eigene-Bank\_Dll\_Assembly.dll ist ein Wrapper vom Bank.dll der das Interface IBankManagement implementiert. Da in der Eigene\_Bank\_DLL\_Assembly.dll ein relativer Pfad auf die Bank.dll verweist muss die Bank.dll sich im Pfad der exe befinden.

Requirements

* Bank.dll
* Eigene\_Bank\_DLL\_Assembly.dll
* Bankanwendung.exe

## Fremde Komponente (XMLController.dll)

Hier sind die Imports vom der XMLController.dll Komponenten enthalten und werden adaptiert, d.h. die Andere-Bank\_DLL\_Assembly.dll ist ein Adapter vom XMLController.dll der das Interface IBankManagement implementiert. Da in der Andere\_Bank\_DLL\_Assembly.dll ein relativer Pfad auf die XMLController.dll verweist muss die XMLController.dll sich im Pfad der exe befinden.

Requirements

* Andere\_Bank\_DLL\_Assembly.dll
* Logging.dll
* XMLController.dll
* Bankanwendung.exe

# Funktionalität der Bankanwendung

In der folgenden Abbildung ist das Flussdiagramm für die Anwendung der Bankanwendung dargestellt. Die Auswahlmöglichkeiten sind das CustomerManagement-Menü und das AccountManagement-Menü.

In den Menüs kann man dann jeweils die Methoden zum Bearbeiten der Konten / Customer aufrufen.

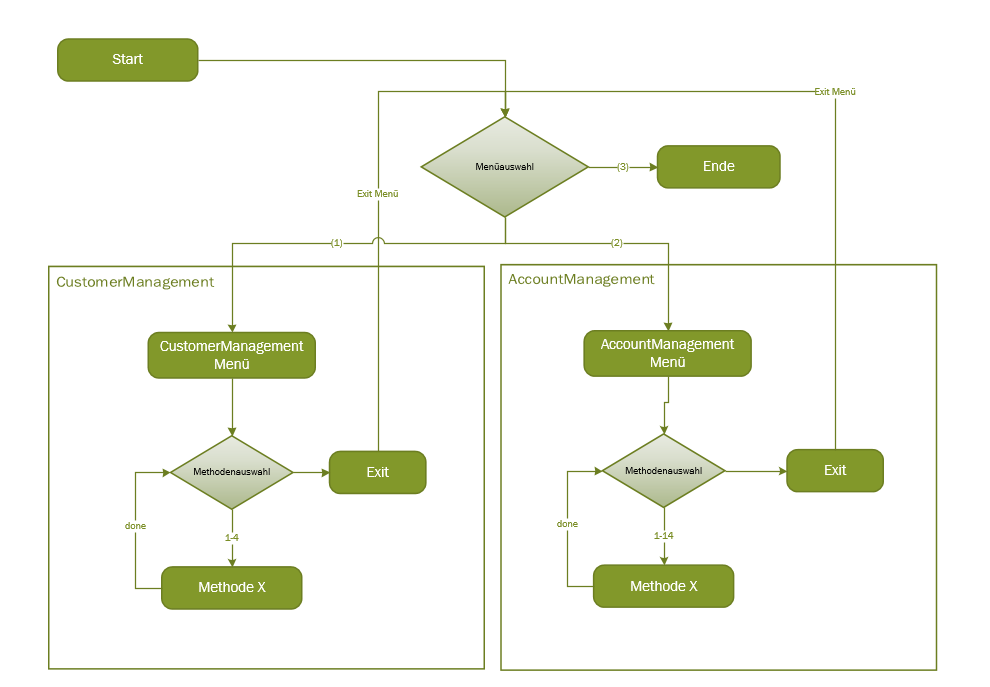


Abbildung 2: Bankanwendung – Flussdiagramm

## Account Management

Hier werden die Interface Methoden für das Account-Management beschrieben.

|  |
| --- |
| int createSavingsAccount(int \_id);  Erstellt ein Sparkonto mit der ID vom Customer |
| int createCreditAccount(int \_id);  Erstellt ein Kreditkonto mit der ID vom Customer |
| void deleteSavingsAccount(int \_sNumber, int \_id);  Löscht das Sparkonto vom User |
| void deleteCreditAccount(int \_cNumber, int \_id);  Löscht das Kreditkonto vom User |
| void depositCreditAcc(int \_cNumber, string \_usage, double \_amount);  Fügt dem KreditKonto Geld hinzu |
| void depositSavingsAcc(int \_sNumber, string \_usage, double \_amount);  Fügt dem SparKonto Geld hinzu |
| void withdrawCreditAcc(int \_cNumber, double \_amount);  Hebt Geld vom KreditKonto ab |
| void transfer(int \_cNumber, int \_toAccNumber, string \_usage, double \_amount);  Hier kann eine Überweisung mit einem Kreditkonto getätigt werden. |
| void addSavingsAccountUser(int \_sNumber, int \_id);  Hier kann ein Verfüger dem SparKonto hinzugefügt werden. |
| void addCreditAccountUser(int \_cNumber, int \_id);  Hier kann ein Verfüger dem KreditKonto hinzugefügt werden. |
| int getBankAccountNumber(int \_id, int \_whichAccount); |

Mit dieser Funktion bekommt man die Kontonummer eines Übergebenen Kontos.

## Customer-Management

Hier werden die Interface Methoden für das Customer-Management beschrieben.

|  |
| --- |
| int createCustomer(string \_Vorname, string \_Nachname, string \_Geburtsdatum,  string \_adresse, string \_Wohnort, string \_Telefon);  Mit createCustomer wird ein neuer Kunde angelegt. |
| void changeCustomer(int \_id, string \_Vorname, string \_Nachname,  string \_address, string \_Wohnort, string \_Telefon);  Diese Funktion ändert die Daten des übergebenen Kunden. |
| int getCustomer(string \_firstName, string \_lastName, string \_birthDate);  Mit getCustomer wird mit Suchparametern der Kunde aus der Datenbank gesucht und dessen ID zurückgegeben. |
| void deleteCustomer(int \_id); |

deleteCustomer löscht einen Kunden.

## Sonstige

Hier werden weitere Interfacemethoden beschrieben.

|  |
| --- |
| void createBankStatement(int \_accNumber);  Erstellt einen Kontoauszug. |
| void convertMoney(int Balance, string \_currency);  Rechnet das Geld in eine andere Währung um. |
| void showChangeOfCourse(int \_cNumber); |

Zeigt die momentane Währung an.

# Zusammenfassung

Es wurde ein C#/.NET Interface entworfen das in den .NET Assemblies verwendet wurde. Bei der DLL vom anderen Team kann man von einem Adapter sprechen wobei es für unseren Teil die Bezeichnung Wrapper zutrifft. So kann die C# Anwendung mit beiden Assemblies arbeiten und es ist nicht ersichtlich dass es sich bei den abhängigen Komponenten um native DLL’s handelt.

Da ein dynamisches Laden der Assemblies nicht nötig war haben wir die nötige Assembly nach Bedarf ein/auskommentiert.

In der Anwendung wurden 2 Menüs geschaffen, das CustomerManagement- und das Accountmanagement Menü in denen die User und die Konten bearbeitet werden können. Der nächste Schritt wird unter anderem noch eine GUI da die Konsolenanwendung sehr umständlich bei der Eingabe ist.