**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«БАРАНОВИЧСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**Факультет** Инженерный

**Кафедра** Информационных систем и технологий

**Дата регистрации работы в деканате \_\_\_\_\_\_\_\_\_**

**Дата регистрации работы на кафедре \_\_\_\_\_\_\_\_\_**

**Отметка о допуске к защите \_\_\_\_\_\_\_\_\_**

**Оценка за защиту \_\_\_\_\_\_\_\_\_**

**КУРСОВАЯ РАБОТА**

**по дисциплине** Объектно-ориентированное программирование

**Тема: «**Разработка программы, моделирующей работу банка, на языке программирования Visual C++»

**Исполнитель:**

Инженерный, 2 курс, ИСТ-21

Галабурда Роман Витольдович

**Руководитель:**

Преподаватель кафедры Информационных систем и технологий

Рогозик Андрей Сергеевич

**Барановичи 2015**

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«БАРАНОВИЧСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

РЕЦЕНЗИЯ

на курсовую работу

(регистрационный №\_\_\_\_\_)

Студента Галабурды Романа Витольдович

Факультет ИНЖЕНЕРНЫЙ

Курс 2

Дисциплина Объектно-ориентированное программирование

Рецензент Рогозик Андрей Сергеевич

Дата получения к/р для рецензирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата возвращения к/р после рецензирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись преподавателя-рецензента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Текст рецензии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc437280154)

[1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc437280155)

[1.1. Постановка задачи..…… 6](#_Toc437280156)

[1.2. Описание объектно-ориентированного подхода 6](#_Toc437280157)

[1.3. Описание предметной области 7](#_Toc437280169)

[2.ПРАКТИЧЕСКАЯ ЧАСТЬ 9](#_Toc437280170)

[2.1 Описание объектов системы, их свойств и методов 9](#_Toc437280171)

[2.2 Результаты тестирования программного продукта 11](#_Toc437280172)

ЗАКЛЮЧЕНИЕ 16

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 17](#_Toc437280177)

[ПРИЛОЖЕНИЕ А 18](#_Toc437280178)

**ВВЕДЕНИЕ**

В современности создание ни одного предприятия, будь то металлургический завод или банк, не обходится без разработки специального комплекса программного обеспечения. Этот комплекс включает в себя кадровую систему, бухгалтерскую систему, клиентскую базу, обработку товаров и услуг, предоставляемых этим предприятием.

Разработка такого предполагает комплекса создание, как минимум, торговой и функциональной логики предприятия, интерфейса для работы персонала с логикой и базы данных для хранения информации, например, о кадрах или продукции.

В рамках данной курсовой работы осуществлена разработка программы, моделирующей работу банка и, соответственно, предоставляющей интерфейс клиентам и добавление, изменения и удаление для них различных услуг, в частности оформление кредитов, банковских пластиковых карт и денежных вкладов (далее депозитов).

Программа разработана на языке Visual C++, позволяющем создавать настольные приложения, имеющие интерфейс Windows Forms в виде окон.

При разработке программы использовался объектно-ориентированный подход к программированию и интегрированная среда разработки Microsoft Visual Studio Ultimate 2012.

Работа с базами данных осуществлена при помощи программы Microsoft Access 2013, входящей в пакет офисных программ Microsoft Office 2013.

Диаграмма построена с помощью веб-сервиса www.draw.io[1], позволяющего создавать различных вида и сложности диаграммы, блок-схемы и пр.

**1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

* 1. **Постановка задачи**

Разработать программу на языке Visual C++, моделирующую работу банка.

* 1. **Описание объектно-ориентированного подхода**

Основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.

Функции объекта, называемые в С++ методами или функциями-членами, обычно предназначены для доступа к данным объекта. Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение. Прямой доступ к данным невозможен, так же, как и их изменение[2].

Объекты представляют собой контейнер, содержащий в себе несколько данных. При этом эти данные могут отличаться между собой по своей структуре.

Когда мы говорим об объектах, мы говорим, что они являются экземплярами классов. Практически все компьютерные языки имеют стандартные типы данных. При помощи классов разработчик может вводить свои типы данных, создавать их экземпляры, которые мы и называем объектами.

Инкапсуляция – свойство объектно-ориентированного подхода, позволяющее использовать компоненты при помощи предоставляемого интерфейса, не задумываясь, что из себя представляет этот компонент. Также инкапсуляция позволяет объединить и защитить жизненно важные для компонента данные.

Инкапсуляция – один из 4 важнейших механизмов объектно-ориентированного программирования (наряду с наследованием, абстракцией и полиморфизмом)[2].

Понятие класса приводит нас к понятию наследования. В повседневной жизни мы часто сталкиваемся с разбиением классов на подклассы: например банк своим клиентам предоставляет услуги, которые делятся на выдачу кредитов, оформление пластиковых банковских карт, осуществление денежных вкладов и т.д.

Принцип, положенный в основу такого деления, заключается в том, что каждый подкласс обладает свойствами, присущими тому классу, из которого выделен данный подкласс.

Базовым называется класс, обобщающий свои подклассы (дочерние или производные классы) и имеющий общие для всех их свойства.

Полиморфизмом называют возможность объектов с одинаковой спецификацией иметь различную реализацию. Например, реализация класса может быть изменена в результате наследования.

Абстракция представляет собой придание объекту характеристик, которые отличают его от всех других объектов, чётко различая границы между ними, и отделение способа использования компонентов от их реализации в виде более простых объектов.

* 1. **Описание предметной области**

Банки своим клиентам предоставляют различные услуги. В нашем случае банк предоставляет оформление кредитов, банковских пластиковых карты и депозитов.

Кредит является разновидностью займа[3]. Но договор кредитования создает определенные права и обязанности для обеих сторон(кредитора и должника) такие, как, например, сроки полного возврата или возврат полученной суммы с некоторой надбавкой в соответствии с процентной ставкой кредита.

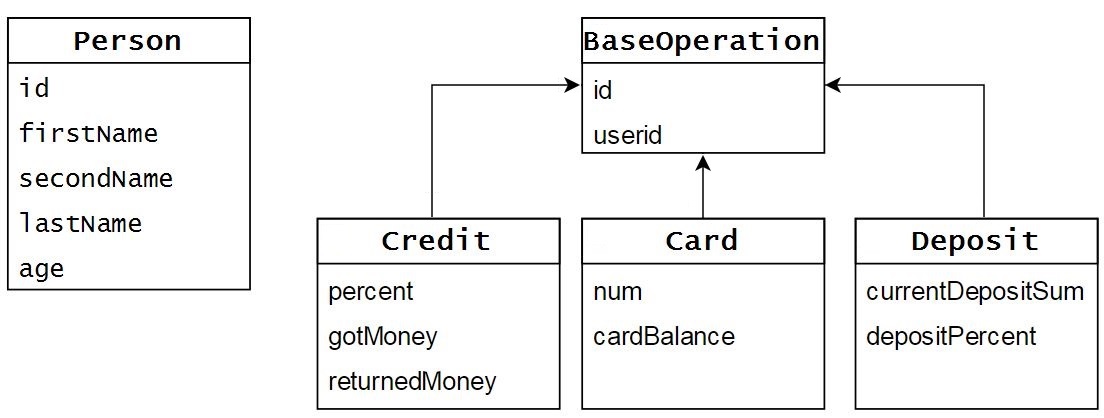
Банковские пластиковые карты являются способом электронной оплаты товаров и услуг. Счёт карты можно пополнять, снимать с него средства.

Депозиты являются сберегательной услугой для клиентов банков. Банк обязуется по истечении срока депозита выдать клиенты полученную сумму с надбавкой в соответствии с процентной ставкой депозита. Процентные ставки зависят от валюты, суммы взноса и срока депозита. Некоторые депозиты имеют функции пополнения, снятия средств тогда, когда другие дают право банку выдать клиенту конечную сумму только по истечении срока депозита.

**2 ПРАКТИЧЕСКАЯ ЧАСТЬ**

**2.1 Описание объектов системы, их свойств и методов**

Программа, разработанная в рамках данной работы и моделирующая работу банка, построена на системе из 5 классов. Схема их наследования изображена на рисунке 2.1.



**Рисунок 2.1 - Схема классов и их наследования.**

Как видно из схемы, класс Person содержит 5 полей, содержащих персональные данные клиента банка: id – идентификационный уникальный номер клиента; firstName, secondName, lastName – фамилия, имя и отчество соответственно; age – возраст. Также содержит 2 перегруженных конструктора, 5 методов доступа к каждому из полей, 1 метод

Класс BaseOperations является базовым абстракным и содержит поля: id - идентификационный уникальный номер операции; userid - идентификационный уникальный номер клиента. Кроме того присутствуют 2 метода для доступа к каждому из полей.

Класс Credit cодержит поля: percent – процент кредита; gotMoney – выданная клиенту сумма; returnedMoney – возвращенная банку сумму. Также класс содержит 2 перегруженных конструктора, 3 метода доступа к каждому из полей, метод getCredit для добавления нового кредит в базу данных, метод payCredit для частичной оплаты кредита, осуществляющий проверку, оплачен ли кредит до конца, и метод changeReturnedMoney, осуществляющие изменения значения возвращенной банку суммы в базе данных.

Класс Card cодержит поля: num – номер банковской карты; cardBalance – баланс карты. Также класс содержит 2 перегруженных конструктора, 2 метода доступа к каждому из полей, метод addCardBalance для инкремента значения баланса экземпляра класса, метод subtractCardBalance для декремента значения баланса экземпляра класса, метод takeCard для создания записи о новой карте в базе данных, fillCard для пополнения баланса, takeMoneyCard для снятия средств с карты.

Класс Deposit cодержит поля: currentDepositSum – текущая сумма депозита; depositPercent – процент кредита. Также класс содержит 2 перегруженных конструктора, 2 метода доступа к каждому из полей, метод addDepositBalance для инкремента значения баланса экземпляра класса, метод subtractDepositBalance для декремента значения баланса экземпляра класса, метод getDeposit для добавления новой записи о депозите в базу данных, метод fillDeposit для пополнения баланса депозита, метод reduceDeposit для частичного снятия средств, closeDeposit для закрытия депозита и полного снятия средств.

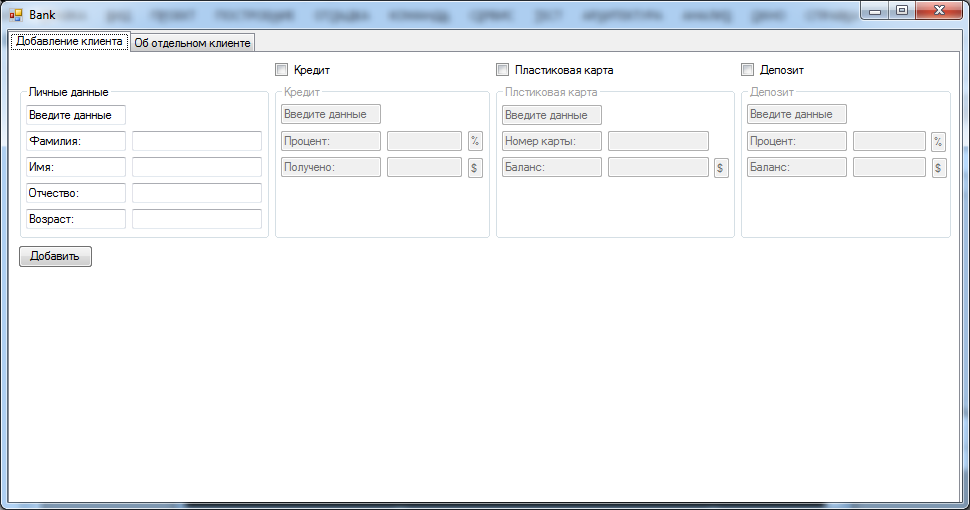
База данных состоит из 4 таблиц TablePerson, TableCredit, TableCard, TableDeposit, содержащих информацию о клиентах, кредитах, пластиковых картах и депозитах соответственно.

База данных подключена в проект программы при помощи компонентов oleDbConnection и oleDbDataAdapter

В Приложении А представлены коды всех файлов.

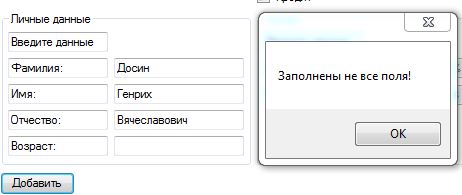
**2.2 Результаты тестирования программного продукта**

Для запуска программы необходимо запустить приложение Bank.exe. В результате откроется форма (рисунок 2.2) с активной вкладкой «Добавление клиента».

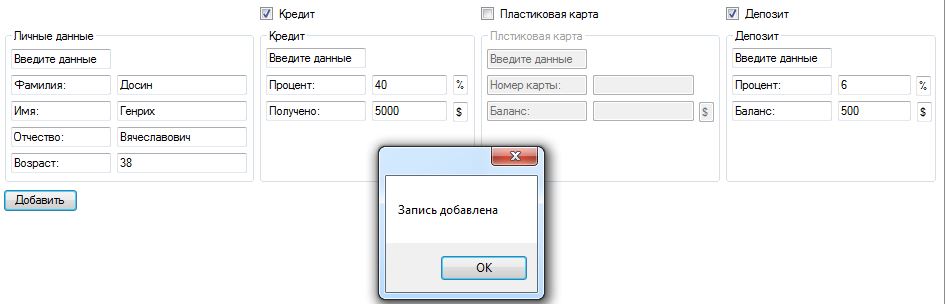


**Рисунок 2.2 - Форма программы.**

# Для добавления нового клиента достаточно заполнить 4 поля его персональными данными и нажать кнопку «Добавить». Если сразу нужно добавить услуги, то для выбранных услуг нужно поставить галочки. В результате этого поля станут доступными для заполнения. Далее необходимо заполнить поля и нажать кнопку «Добавить». Тест этой функции изображен на рисунках 2.3-2.4.

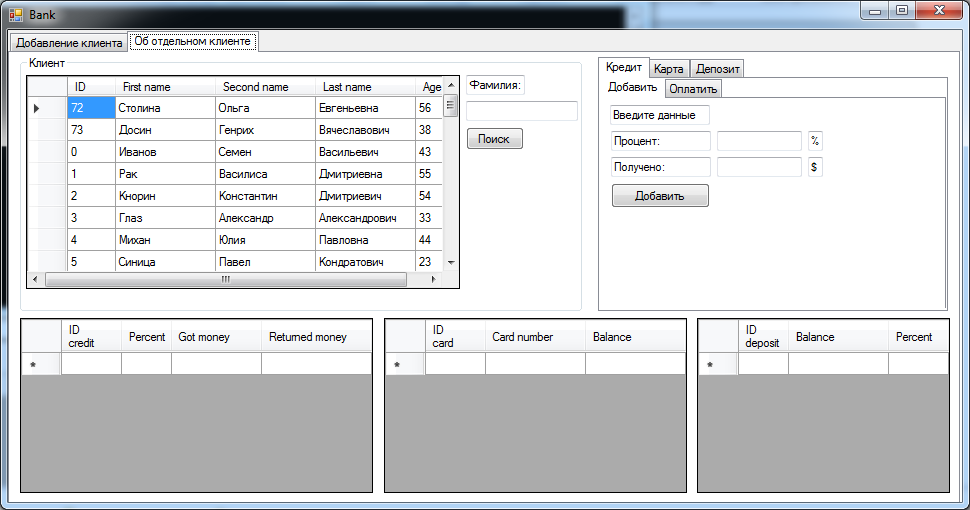
****

**Рисунок 2.3 - Защита от заполнения не всех полей.**

****

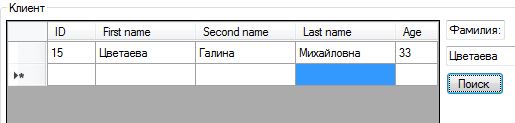
**Рисунок 2.4 - Добавления записи о клиенте одновременно с добавлением кредита и депозита.**

Далее перейдем на вторую вкладку под названием «Об отдельном клиенте». При этом каждый раз происходит обновление таблицы клиентов данными из таблицы TablePerson базы данных BankClient.accdb, что изображено на рисунке 2.5.

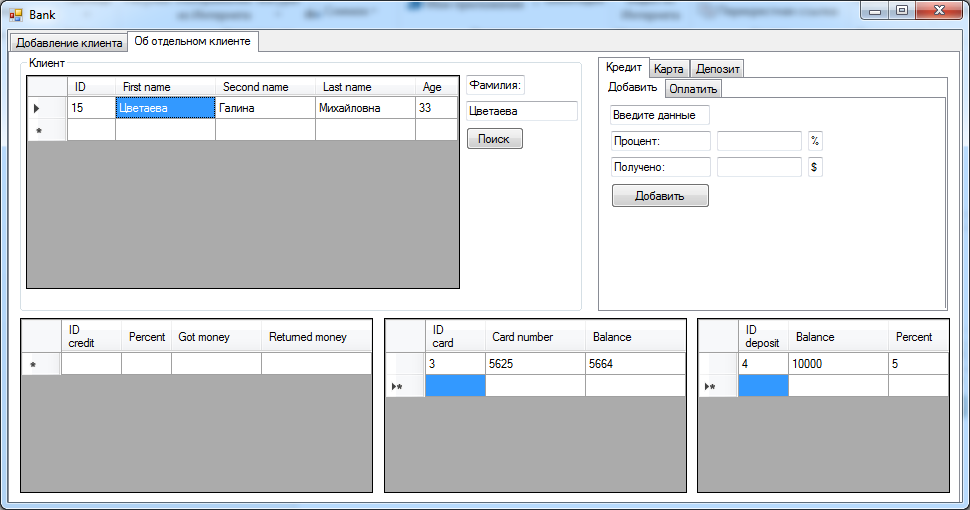


**Рисунок 2.5 - Вкладка «Об отдельном клиенте».**

Выполнить поиск по фамилии можно осуществив ввод искомой фамилии справа от таблицы и нажав кнопку «Поиск», что изображено на рисунке 2.6.

****

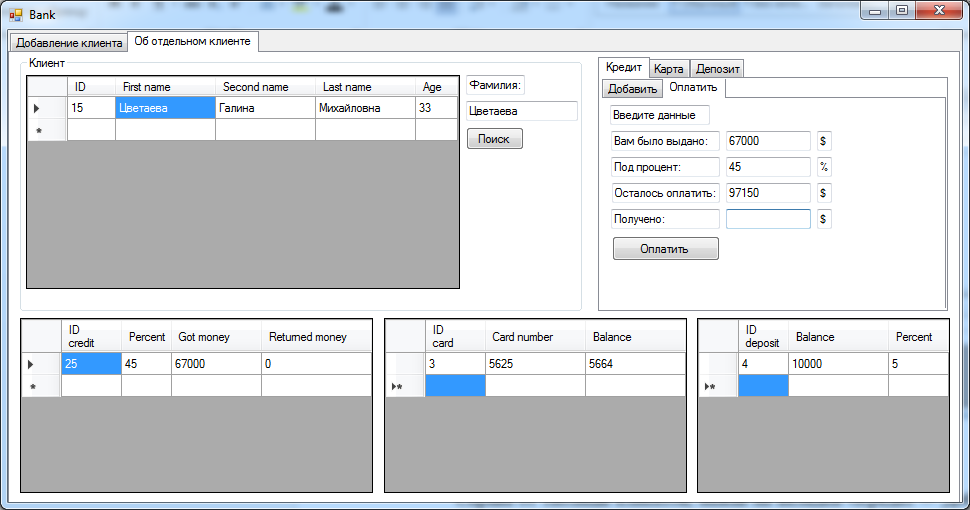
**Рисунок 2.6 - Поиск по фамилии.**

При нажатии на строку таблицы с клиентами заполнятся таблицы кредитов, карт и депозитов соответствующими этому клиенту данными, что изображено на рисунке 2.7. 

**Рисунок 2.7 - Вывод всех услуг для выбранного клиента.**

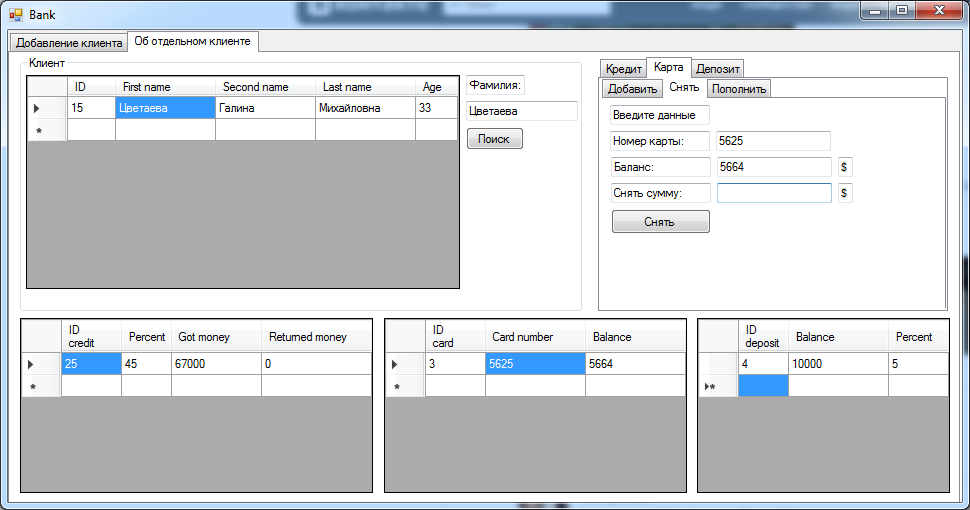
Справа от таблицы клиентов, нажав на вкладки «Кредит -> Добавить», «Карта -> Добавить» или «Депозит -> Добавить», можно добавить соответствующие услуги для текущего клиента.

Нажав на один из кредитов в соответствующей таблице, вам будет предложено оплатить его, что изображено на рисунке 2.8. Для этого необходимо заполнить поля и нажать кнопку «Оплатить». При этом автоматический выполнится проверка, полностью ли оплачен кредит, и, если это так, запись о данном кредите удалится из базы данных.



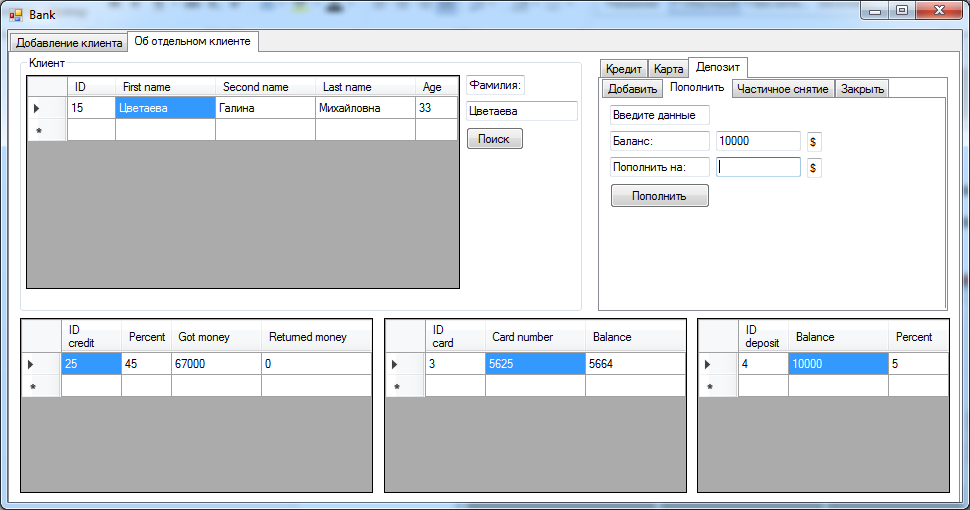
**Рисунок 2.8 - Информация о выбранном кредите.**

Нажав на одну из карт в соответствующей таблице, вам будет предложено оплатить его, что изображено на рисунке 2.9. Для пополнения баланса карты следует переключиться на вкладку «Пополнить», заполнить необходимые поля и нажать кнопку «Пополнить».



**Рисунок 2.9 - Информация о выбранной карте.**

Нажав на один из депозитов в соответствующей таблице, вам будет предложено пополнить его, что изображено на рисунке 2.10. Для частичного снятия баланса депозита следует переключиться на вкладку «Частичное снятие». Для полного снятия баланса и закрытия депозита следует выбрать вкладку «Закрыть» и нажать кнопку «Закрыть».



**Рисунок 2.10 - Информация о выбранном депозите.**

**ЗАКЛЮЧЕНИЕ**

Данная работа показывает, что процесс разработки программного обеспечения для предприятия требует немалого количества знаний о языке программирования Visual С++, кроме того, и о базах данных.

Кроме того, важными моментами при разработке программы являются описание спецификации программы, алгоритма ее реализации, хотя Visual C++ и предоставляет необходимые средства для упрощения процесса программирования.

Проведенное тестирование программы позволяет сделать вывод о работоспособности программы и ее соответствии заданию. Как и любое решение в любой области наук, промышленности или социальной жизни, так и любую программу можно усовершенствовать. То есть дополнять новыми возможностями, делать интерфейс более интерактивным и дружелюбным, разрабатывать дизайн приложения и т.д. С увеличением опыта и навыков программист может разрабатывать приложение намного сложнее и объемнее.

А так как области, в которых данное приложение может найти применение, развиваются все больше и больше, то разработка подобных приложений все более востребована, а тем более разработка объемных в плане своих возможностей приложений.

В результате проделанной работы я получил опыт разработки приложений на языке Visual C++, научился пользоваться интегрированной средой разработки Microsoft Visual Studio Ultimate 2012 и интегрировать работу с базами данных Microsoft Access 2013 в проект.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Flow Chart Maker & Online Diagram Software [Электронный ресурс]. – 2015. Режим доступа: https://www.draw.io/. – Дата доступа: 14.12.2015.
2. Р. Лафоре, Объектно-ориентированное программирование в С++ / Р. Лафоре. - 4-е издание. Москва: 2004. – 922 с.
3. Шляпников А.В, Кредиты и займы / Шляпников А.В. – 25-й выпуск. Москва: 2006. – 51 с.

# ПРИЛОЖЕНИЕ А

Текст файла Person.h

#pragma once

using namespace System::Data::OleDb;

ref class Person

{

private:

System::UInt64 ^id;

System::String ^firstName;

System::String ^secondName;

System::String ^lastName;

System::UInt16 ^age;

public:

Person(System::String ^fn, System::String ^sn, System::String ^ln, System::UInt16 ^age);

Person(System::UInt64 ^id, System::String ^fn, System::String ^sn, System::String ^ln, System::UInt16 ^age);

System::UInt64 ^getid() { return this->id; }

System::String ^getFirstName() { return this->firstName; }

System::String ^getSecondName() { return this->secondName; }

System::String ^getLastName() { return this->lastName; }

System::UInt16 ^getAge() {return this->age; };

System::Void setid(OleDbConnection ^odbc);

void DBAddPerson(OleDbConnection ^odbc);

};

Текст файла Person.cpp

#include "Person.h"

#include "BaseOperation.h"

#include "Credit.h"

#include "Deposit.h"

#include "Card.h"

Person::Person(System::String ^fn, System::String ^sn, System::String ^ln, System::UInt16 ^age)

{

this->firstName = fn;

this->secondName = sn;

this->lastName = ln;

this->age = age;

}

Person::Person(System::UInt64 ^id, System::String ^fn, System::String ^sn, System::String ^ln, System::UInt16 ^age)

{

this->id = id;

this->firstName = fn;

this->secondName = sn;

this->lastName = ln;

this->age = age;

}

System::Void Person::setid(OleDbConnection ^odbc)

{

odbc->Open();

System::String ^myQuery = "SELECT personid FROM TablePerson WHERE fname='" + this->getFirstName() + "'";

OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery,odbc);

OleDbDataReader ^myReader = myCommand->ExecuteReader();

try

{

while(myReader->Read())

this->id = System::Convert::ToUInt64(myReader["personid"]);

}

\_\_finally { myReader->Close(); }

odbc->Close();

}

System::Void Person::DBAddPerson(OleDbConnection ^odbc)

{

odbc->Open();

System::String ^myQuery = "INSERT INTO TablePerson (fname,sname,lname,age) VALUES(@fname,@sname,@lname,@age)";

OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->Parameters->AddWithValue("@fname", this->getFirstName());

myCommand->Parameters->AddWithValue("@sname", this->getSecondName());

myCommand->Parameters->AddWithValue("@lname", this->getLastName());

myCommand->Parameters->AddWithValue("@age", this->getAge());

myCommand->ExecuteNonQuery();

odbc->Close();

}

Текст файла BaseOperations.h

#pragma once

#include "Person.h"

ref class BaseOperation

{

public:

// fields

System::UInt64 ^id;

System::UInt64 ^userid;

// methods

System::UInt64 ^getid() { return this->id; }

System::UInt64 ^getuserid() { return this->userid; }

};

Текст файла Credit.h

#pragma once

#include "BaseOperation.h"

ref class Credit : public BaseOperation

{

private:

System::UInt16 ^percent;

System::UInt64 ^gotMoney;

System::UInt64 ^returnedMoney;

public:

Credit(System::UInt64 ^userid, System::UInt16 ^percent, System::UInt64 ^gotMoney, System::UInt64 ^returnedMoney);

Credit(System::UInt64 ^id, System::UInt64 ^userid, System::UInt16 ^percent, System::UInt64 ^gotMoney, System::UInt64 ^returnedMoney);

System::UInt16 ^getPercent() { return percent; }

System::UInt64 ^getGotMoney() { return gotMoney; }

System::UInt64 ^getReturnedMoney() { return returnedMoney; }

System::Void changeReturnedMoney(System::UInt64 sum);

System::Void getCredit(OleDbConnection ^odbc);

System::Void payCredit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 creditid, System::UInt64 need\_pay);

};

Текст файла Credit.cpp

#include "Credit.h"

Credit::Credit(System::UInt64 ^userid, System::UInt16 ^percent, System::UInt64 ^gotMoney, System::UInt64 ^returnedMoney)

{

this->userid = userid;

this->percent = percent;

this->gotMoney = gotMoney;

this->returnedMoney = returnedMoney;

}

Credit::Credit(System::UInt64 ^id, System::UInt64 ^userid, System::UInt16 ^percent, System::UInt64 ^gotMoney, System::UInt64 ^returnedMoney)

{

this->id = id;

this->userid = userid;

this->percent = percent;

this->gotMoney = gotMoney;

this->returnedMoney = returnedMoney;

}

System::Void Credit::changeReturnedMoney(System::UInt64 sum)

{

\*this->returnedMoney = \*this->returnedMoney + sum;

}

System::Void Credit::getCredit(OleDbConnection ^odbc)

{

odbc->Open();

System::String ^myQuery = "INSERT INTO [TableCredit] ([userid],[percent],[gotMoney],[returnedMoney]) VALUES(@userid,@percent,@gotMoney,@returnedMoney)";

OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->Parameters->AddWithValue("@userid",this->getuserid());

myCommand->Parameters->AddWithValue("@percent",System::Convert::ToString(this->getPercent()));

myCommand->Parameters->AddWithValue("@gotMoney",System::Convert::ToString(this->getGotMoney()));

myCommand->Parameters->AddWithValue("@returnedMoney",System::Convert::ToString(0));

myCommand->ExecuteNonQuery();

odbc->Close();

}

System::Void Credit::payCredit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 creditid, System::UInt64 need\_pay)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of returned money

if((tb->Text != ""))

{

this->changeReturnedMoney(System::Convert::ToUInt64(tb->Text));

// check credit status

if(\*this->getReturnedMoney() >= need\_pay)

{

myQuery = "DELETE FROM [TableCredit] WHERE [creditid] = " + creditid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

else

{

myQuery = "UPDATE [TableCredit] SET [returnedMoney] = " + System::Convert::ToString(this->getReturnedMoney()) + " WHERE [creditid] = " + creditid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}

Текст файла Card.h

#pragma once

#include "BaseOperation.h"

ref class Card : public BaseOperation

{

private:

System::UInt64 ^num;

System::UInt64 ^cardBalance;

public:

Card(System::UInt64 ^userid, System::UInt64 ^num, System::UInt64 ^cardBalance);

Card(System::UInt64 ^id, System::UInt64 ^userid, System::UInt64 ^num, System::UInt64 ^cardBalance);

System::UInt64 ^getNum() { return num; }

System::UInt64 ^getCardBalance() { return cardBalance; }

System::Void addCardBalance(System::UInt64 sum);

System::Void subtractCardBalance(System::UInt64 sum);

System::Void takeCard(OleDbConnection ^odbc);

System::Void fillCard(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 cardid);

System::Void takeMoneyCard(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 cardid);

};

Текст файла Card.cpp

#include "Card.h"

Card::Card(System::UInt64 ^userid, System::UInt64 ^num, System::UInt64 ^cardBalance)

{

this->userid = userid;

this->num = num;

this->cardBalance = cardBalance;

}

Card::Card(System::UInt64 ^id, System::UInt64 ^userid, System::UInt64 ^num, System::UInt64 ^cardBalance)

{

this->id = id;

this->userid = userid;

this->num = num;

this->cardBalance = cardBalance;

}

System::Void Card::addCardBalance(System::UInt64 sum)

{

\*this->cardBalance = \*this->cardBalance + sum;

}

System::Void Card::subtractCardBalance(System::UInt64 sum)

{

\*this->cardBalance = \*this->cardBalance - sum;

}

System::Void Card::takeCard(OleDbConnection ^odbc)

{

odbc->Open();

System::String ^myQuery = "INSERT INTO TableCard (userid,cardNumber,cardBalance) VALUES(@userid,@cardNumber,@cardBalance)";

OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->Parameters->AddWithValue("@userid",this->getuserid());

myCommand->Parameters->AddWithValue("@cardNumber", this->getNum());

myCommand->Parameters->AddWithValue("@cardBalance", this->getCardBalance());

myCommand->ExecuteNonQuery();

odbc->Close();

}

System::Void Card::fillCard(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 cardid)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of card balance

if(tb->Text != "")

{

this->addCardBalance(System::Convert::ToUInt64(tb->Text));

myQuery = "UPDATE [TableCard] SET [cardBalance] = " + System::Convert::ToString(this->cardBalance) + " WHERE [cardid] = " + cardid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}

System::Void Card::takeMoneyCard(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 cardid)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of card balance

if((tb->Text != "") && (System::Convert::ToUInt64(tb->Text) <= \*this->cardBalance))

{

this->subtractCardBalance(System::Convert::ToUInt64(tb->Text));

myQuery = "UPDATE [TableCard] SET [cardBalance] = " + System::Convert::ToString(this->cardBalance) + " WHERE [cardid] = " + cardid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}

Текст файла Deposit.h

#pragma once

#include "BaseOperation.h"

ref class Deposit : public BaseOperation

{

private:

System::UInt64 ^currentDepositSumm;

System::UInt16 ^depositPercent;

public:

Deposit(System::UInt64 ^userid, System::UInt64 ^sum, System::UInt16 ^depPercent);

Deposit(System::UInt64 ^id, System::UInt64 ^userid, System::UInt64 ^sum, System::UInt16 ^depPercent);

System::UInt64 ^getCurrentDepositSumm() { return \*this->currentDepositSumm; }

System::UInt16 ^getDepositPercent() { return \*this->depositPercent; }

System::Void addDepositBalance(System::UInt64 sum);

System::Void subtractDepositBalance(System::UInt64 sum);

System::Void getDeposit(OleDbConnection ^odbc, System::UInt64 ^userid);

System::Void fillDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid);

System::Void reduceDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid);

System::Void closeDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid);

};

Текст файла Deposit.cpp

#include "Deposit.h"

Deposit::Deposit(System::UInt64 ^userid, System::UInt64 ^sum, System::UInt16 ^depPercent)

{

this->userid = userid;

this->currentDepositSumm = sum;

this->depositPercent = depPercent;

}

Deposit::Deposit(System::UInt64 ^id, System::UInt64 ^userid, System::UInt64 ^sum, System::UInt16 ^depPercent)

{

this->id = id;

this->userid = userid;

this->currentDepositSumm = sum;

this->depositPercent = depPercent;

}

System::Void Deposit::addDepositBalance(System::UInt64 sum)

{

\*this->currentDepositSumm = \*this->currentDepositSumm + sum;

}

System::Void Deposit::subtractDepositBalance(System::UInt64 sum)

{

\*this->currentDepositSumm = \*this->currentDepositSumm - sum;

}

System::Void Deposit::getDeposit(OleDbConnection ^odbc, System::UInt64 ^userid)

{

odbc->Open();

System::String ^myQuery = "INSERT INTO TableDeposit (userid,currentDepositSumm,depositPercent) VALUES(@userid,@currentDepositSumm,@depositPercent)";

OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->Parameters->AddWithValue("@userid", userid);

myCommand->Parameters->AddWithValue("@currentDepositSumm", this->getCurrentDepositSumm());

myCommand->Parameters->AddWithValue("@depositPercent", this->getDepositPercent());

myCommand->ExecuteNonQuery();

odbc->Close();

delete myCommand;

delete myQuery;

}

System::Void Deposit::fillDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of card balance

if((tb->Text != "") && (System::Convert::ToUInt64(tb->Text) <= \*this->currentDepositSumm))

{

this->addDepositBalance(System::Convert::ToUInt64(tb->Text));

myQuery = "UPDATE [TableDeposit] SET [currentDepositSumm] = " + System::Convert::ToString(this->currentDepositSumm) + " WHERE [depositid] = " + depositid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}

System::Void Deposit::reduceDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of card balance

if((tb->Text != "") && (System::Convert::ToUInt64(tb->Text) <= \*this->currentDepositSumm))

{

this->subtractDepositBalance(System::Convert::ToUInt64(tb->Text));

myQuery = "UPDATE [TableDeposit] SET [currentDepositSumm] = " + System::Convert::ToString(this->currentDepositSumm) + " WHERE [depositid] = " + depositid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}

System::Void Deposit::closeDeposit(OleDbConnection ^odbc, System::Windows::Forms::TextBox ^tb, System::UInt64 depositid)

{

System::String ^myQuery = nullptr;

OleDbCommand ^myCommand = nullptr;

// change value of card balance

if(tb->Text != "")

{

myQuery = "DELETE FROM [TableDeposit] WHERE [depositid] = " + depositid;

myCommand = gcnew OleDbCommand(myQuery,odbc);

myCommand->ExecuteNonQuery();

System::Windows::Forms::MessageBox::Show("Депозит закрыт. К выдаче: " + tb->Text + "$.");

}

else

System::Windows::Forms::MessageBox::Show("Введите значение!");

}