**main.cpp**

#include <iostream>

#include <fstream>

#include <thread>

#include "Windows.h"

#include "Interface.h"

using namespace std;

int main()

{

system("color 70");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// запрет аварийного закрытия окна

HWND hWnd = GetForegroundWindow();

EnableWindow(hWnd, false);

List<Deposit> depositList;

File<Deposit>::FileOpenDeposit(depositList, "deposit.txt");

for (int i = 0; i < depositList.size(); i++)

{

thread\* Thr = new thread(updateBalance, &depositList[i]);

Thr->detach();

}

List<Loan> loanList;

File<Loan>::FileOpenLoan(loanList, "loan.txt");

for (int i = 0; i < loanList.size(); i++)

{

thread\* Thr = new thread(updateBalance, &loanList[i]);

Thr->detach();

}

while (true)

{

cout << "Вас приветствует банковская система!" << endl;

cout << "Для входа введите имя пользователя или введите '0' для выхода: " << endl;

List<Account> AccountList;

File<Account>::FileOpenAccount(AccountList, "account.txt");

Account user;

string username;

rewind(stdin);

cin >> username;

if (username == "0") break;

user = AccountList.getUser(username);

cout << "Добро пожаловать " << username << endl;

system("pause");

Interface<int>::menu(user, depositList, loanList, AccountList);

File<Account>::FileSaveAccount(AccountList, "account.txt");

}

File<Deposit>::FileSaveDeposit(depositList, "deposit.txt");

File<Loan>::FileSaveLoan(loanList, "loan.txt");

return 0;

}

**Account.h**

#pragma once

#include <string>

#include "InputException.h"

using namespace std;

class Account

{

protected:

string username;

double balance;

public:

explicit Account(string \_username = "", double \_balance = 0)

: username(\_username), balance(\_balance) {};

~Account() {};

const string &getUsername() const;

double getBalance() { return this->balance; }

void setUsername(const string & tmp) { this->username = tmp; }

void setBalance(double \_balance) { this->balance = \_balance; }

string InputCharCheck(string input);

bool CheckCorrect(string input, bool fl) throw (InputException);

friend void operator >> (std::istream & is, Account & tmp);

friend void operator << (ostream & os, Account & tmp);

};

bool CheckMoney(double money);

bool CheckList(double num);

int CheckInput(double input1, double input2);

**Account.cpp**

#pragma once

#include <iomanip>

#include "Account.h"

#include "ListException.h"

using namespace std;

const string &Account::getUsername() const

{

return username;

}

void operator << (std::ostream & os, Account & tmp)

{

os << setw(15) << tmp.username << setw(12) << tmp.balance;

}

void operator >> (std::istream & is, Account &tmp)

{

tmp.username = tmp.InputCharCheck(tmp.username);

}

bool Account::CheckCorrect(string input, bool fl) throw (InputException)

{

int len = input.length();

for (int i = 0; i < len; i++)

{

if (input[i] < 'A' || input[i] > 'Z')

{

if (input[i] < 'a' || input[i] > 'z')

{

if (input[i] < '0' || input[i] > '9')

{

fl = false;

throw InputException(1);

}

}

}

}

if (len > 20)

{

fl = false;

throw InputException(3);

}

if (input[0] >= '0' && input[0] <= '9')

{

fl = false;

throw InputException(2);

}

return fl;

}

string Account::InputCharCheck(string input)

{

bool fl = true;

do

{

try

{

Account tmp;

fflush(stdin);

cin.sync();

cin.clear();

cin >> input;

tmp.CheckCorrect(input, fl);

}

catch (InputException e)

{

e.ErrorText();

cin.clear();

cin.sync();

InputCharCheck(input);

}

fflush(stdin);

} while (!fl);

return input;

}

bool CheckMoney(double money)

{

try

{

if (money < 0)

throw InputException(5);

}

catch (InputException e)

{

e.ErrorText();

return true;

}

return false;

}

int CheckInput(double input1, double input2)

{

bool flag = true;

int tmp = 0;

do

{

try

{

cin.sync();

cin.clear();

rewind(stdin);

cin >> tmp;

flag = true;

if (!cin || cin.peek() != '\n')

{

flag = false;

throw InputException(4);

}

if (tmp < input1 || tmp > input2)

{

flag = false;

throw 5;

}

}

catch (InputException e)

{

e.ErrorText();

cin.sync();

cin.clear();

rewind(stdin);

}

catch (...)

{

cin.sync();

cin.clear();

rewind(stdin);

cout << "Введите число в интервале от " << input1 << " до " << input2 << endl;

}

} while (!flag);

return tmp;

}

bool CheckList(double num)

{

try

{

if (!num) throw ListException(1);

}

catch (ListException e)

{

e.ErrorText();

return true;

}

return false;

}

**Deposit.h**

#pragma once

#include "Account.h"

class Deposit : public Account

{

int percent;

public:

explicit Deposit(const string &username = "", double balance = 0, int percent = 5);

int getPercent() const;

void setPercent(int percent);

friend void updateBalance(Deposit\* deposit);

friend void operator >> (istream & is, Deposit & tmp);

friend void operator << (ostream & os, Deposit & tmp);

~Deposit() {};

};

**Deposit.cpp**

#include <iostream>

#include <thread>

#include "Windows.h"

#include "Deposit.h"

Deposit::Deposit(const string &username, double balance, int percent)

: Account(username, balance), percent(percent) {}

int Deposit::getPercent() const

{

return percent;

}

void Deposit::setPercent(int percent)

{

Deposit::percent = percent;

}

void operator >> (std::istream & is, Deposit & tmp)

{

cout << "Введите сумму депозита: ";

tmp.balance = CheckInput(0, 999999);

cout << "Введите процент депозита (1% - 10%): ";

tmp.percent = CheckInput(1, 10);

}

void operator << (std::ostream & os, Deposit & tmp)

{

os << tmp.balance << endl;

cout << "-------------------------------" << endl;

os << "Процентная ставка: " << tmp.percent << "%" << endl;

}

void updateBalance(Deposit\* deposit)

{

while (true)

{

Sleep(60000);

deposit->balance = deposit->balance \* (100 + deposit->percent) / 100;

}

}

**MoneyTransfer.h**

#pragma once

#include "Account.h"

class MoneyTransfer : public Account

{

protected:

string partnerUsername;

public:

explicit MoneyTransfer(string \_username = "", double \_balance = 0, string partnerUsername = "")

: Account(\_username, \_balance), partnerUsername(partnerUsername) {};

~MoneyTransfer() {}

const string &getPartnerUsername() const;

void setPartnerUsername(const string &partnerUsername);

friend void operator >> (istream & is, MoneyTransfer & tmp);

friend void operator << (ostream & os, MoneyTransfer & tmp);

friend bool operator == (MoneyTransfer tmp1, MoneyTransfer tmp2);

void operator = (const MoneyTransfer & tmp);

};

**MoneyTransfer.cpp**

#include <iomanip>

#include "MoneyTransfer.h"

#include "InputException.h"

using namespace std;

void operator >> (std::istream & is, MoneyTransfer & tmp)

{

cout << "Введите сумму: ";

tmp.balance = CheckInput(0, 1000000);

}

void operator << (std::ostream & os, MoneyTransfer & tmp)

{

os << setw(12) << tmp.balance;

}

void MoneyTransfer::operator = (const MoneyTransfer & tmp)

{

balance = tmp.balance;

partnerUsername = tmp.partnerUsername;

}

bool operator== (MoneyTransfer tmp1, MoneyTransfer tmp2)

{

return tmp1.balance == tmp2.balance && tmp1.partnerUsername == tmp2.partnerUsername;

}

const string &MoneyTransfer::getPartnerUsername() const

{

return partnerUsername;

}

void MoneyTransfer::setPartnerUsername(const string &partnerUsername)

{

MoneyTransfer::partnerUsername = partnerUsername;

}

**Loan.h**

#pragma once

#include "Deposit.h"

class Loan : public Deposit

{

private:

string partnerUsername;

public:

explicit Loan(const string &username = "", double balance = 0, int percent = 0, const string &partnerUsername = "") :

Deposit(username, balance, percent), partnerUsername(partnerUsername) {};

~Loan() {};

string getPartnerUsername() const { return this->partnerUsername; }

void setPartnerUsername(string tmp) { this->partnerUsername = tmp; }

friend void operator >> (istream & is, Loan & tmp);

friend void operator << (ostream & os, Loan & tmp);

};

**Loan.cpp**

#pragma once

#include "Deposit.h"

class Loan : public Deposit

{

private:

string partnerUsername;

public:

explicit Loan(const string &username = "", double balance = 0, int percent = 0, const string &partnerUsername = "") :

Deposit(username, balance, percent), partnerUsername(partnerUsername) {};

~Loan() {};

string getPartnerUsername() const { return this->partnerUsername; }

void setPartnerUsername(string tmp) { this->partnerUsername = tmp; }

friend void operator >> (istream & is, Loan & tmp);

friend void operator << (ostream & os, Loan & tmp);

};

**Exception.h**

#pragma once

#include <iostream>

using namespace std;

class Exception

{

protected:

int error;

public:

Exception() { error = 0; }

Exception(int \_error) { error = \_error; }

virtual void ErrorText() {};

~Exception() {};

};

**InputException.h**

#pragma once

#include "Exception.h"

class InputException : public Exception

{

public:

InputException() : Exception() {};

InputException(int \_error) : Exception(\_error) {};

void ErrorText();

~InputException(){};

};

**InputException.cpp**

#include <iostream>

#include "InputException.h"

using namespace std;

void InputException::ErrorText()

{

switch (error)

{

case 2:

cout << "Ошибка 1.2" << endl;

cout << "Имя пользователя не может начинаться с цифры!" << endl << endl;

break;

case 3:

cout << "Ошибка 1.3" << endl;

cout << "Превышен лимит по вводу символов!" << endl;

cout << "Максимально допустимое значение - 20 символов." << endl << endl;

break;

case 4:

cout << "Ошибка 1.4" << endl;

cout << "Вы ввели символы вместо числа либо превышен предел по вводу чисел!" << endl << endl;

break;

case 5:

cout << "Ошибка 1.5" << endl;

cout << "Невозможно совершить операцию!" << endl;

cout << "Баланс пользователя не может быть отрицательным" << endl;

break;

default:

cout << "Ошибка ввода" << endl;

cout << "Повторите ввод" << endl;

break;

}

}

**ListException.h**

#pragma once

#include "Exception.h"

class ListException : public Exception

{

public:

explicit ListException(int \_error = 0) : Exception(\_error) {};

void ErrorText();

~ListException() {};

};

**ListException.cpp**

#include <iostream>

#include "ListException.h"

using namespace std;

void ListException::ErrorText()

{

switch (error)

{

case 1:

{

cout << "Ошибка 2.1:" << endl;

cout << "Список пуст!" << endl;

break;

}

}

}

**List.h**

#pragma once

#include <iostream>

#include <iomanip>

#include "Deposit.h"

#include "MoneyTransfer.h"

using namespace std;

// узел для контейнера список

template<typename T>

struct Node

{

T data;

Node<T> \*next;

Node<T> \*prev;

};

// класс итератора

template <typename T>

class Iterator

{

Node<T> \*current;

public:

Iterator()

{

current = nullptr;

}

Iterator(Iterator<T> &iter)

{

current = iter.current;

}

Iterator(Node<T> \*node)

{

current = node;

}

~Iterator() {}

// получение узла из итератора

Node<T> \*get\_node()

{

return current;

}

// сдвиг итератора

bool operator++(int i)

{

if (current == nullptr) return false;

current = current->next;

return true;

}

bool operator--(int i)

{

if (current->prev == nullptr) return false;

current = current->prev;

return true;

}

// получение значения из итератора

T &operator\*()

{

return current->data;

}

// операторы сравнения для итератора

bool operator==(Iterator &iter)

{

if (current == nullptr && iter.current == nullptr)

return true;

if (current == nullptr || iter.current == nullptr)

return false;

return current->data == iter.current->data && current->next == iter.current->next &&

current->prev == iter.current->prev;

}

bool operator!=(Iterator &iter)

{

if (current == nullptr && iter.current == nullptr)

return false;

if ((current == nullptr && iter.current != nullptr) || (current != nullptr && iter.current == nullptr))

return true;

return !(current->data == iter.current->data && current->next == iter.current->next &&

current->prev == iter.current->prev);

}

};

// класс-контейнер cписок

template<typename T>

class List

{

protected:

friend class Iterator<T>;

Node<T> \*head;

Node<T> \*tail;

long amount;

public:

List()

{

head = nullptr;

tail = nullptr;

amount = 0;

}

~List()

{

while (head)

{

this->pop\_head();

}

}

// длина списка

long size()

{

return this->amount;

}

// добавление с головы

void push\_head(T input\_object)

{

if (head == nullptr)

{

head = new Node<T>;

head->data = input\_object;

head->next = nullptr;

head->prev = nullptr;

tail = head;

amount++;

return;

}

Node<T> \*node = new Node<T>;

node->data = input\_object;

node->next = head;

node->prev = nullptr;

head->prev = node;

head = node;

amount++;

return;

}

// добавление в хвост

void push\_tail(T input\_object)

{

if (head == nullptr)

{

head = new Node<T>;

head->data = input\_object;

head->next = nullptr;

head->prev = nullptr;

tail = head;

amount++;

return;

}

else

{

Node<T> \*node = new Node<T>;

node->data = input\_object;

node->next = nullptr;

node->prev = tail;

tail->next = node;

tail = node;

amount++;

return;

}

}

// удаление из головы

T pop\_head()

{

if (!(head)) return T();

T data = head->data;

Node<T> \*node = head;

head = head->next;

if (head)

head->prev = nullptr;

delete node;

amount--;

return data;

}

// удаление из хвоста

T pop\_tail()

{

if (!(head)) return T();

T data = tail->data;

Node<T> \*node = tail;

if (tail != head) {

tail = tail->prev;

tail->next = nullptr;

}

else {

head = tail = nullptr;

}

delete node;

amount--;

return data;

}

T pop(int num)

{

T data = this->operator[](num);

Node<T> \*curr = head;

for (long i = 0; i < num; i++)

curr = curr->next;

Node<T> \*next = curr->next;

Node<T> \*prev = curr->prev;

next->prev = prev;

prev->next = next;

delete curr;

amount--;

return data;

}

// произвольный доступ к объектам

T &operator[](long num)

{

Node<T> \*curr = head;

if (num < 0 || num >= amount) return curr->data;

for (long i = 0; i < num; i++)

curr = curr->next;

return curr->data;

}

// функции для работы с итератором

Node<T> \*begin()

{

return head;

}

Node<T> \*end()

{

if (tail != nullptr) return tail->next;

return tail;

}

// вывод на экран содержимого списка

void output()

{

cout << "-------------------------------" << endl;

cout << setw(3) << "#" << setw(15) << "Аккаунт" << setw(12) << "Сумма" << endl;

int i = 0;

if (!head) cout << "Список пуст!";

else

for (Node<T> \*node = head; node != nullptr; node = node->next)

{

cout << "-------------------------------" << endl;

cout << setw(3) << i + 1 << node->data;

cout << endl;

i++;

}

cout << "-------------------------------" << endl;

cout << endl;

}

// вывод на экран информации о займах

void output\_loans()

{

cout << "-------------------------------" << endl;

cout << setw(3) << "#" << setw(12) << "Сумма" << endl;

int i = 0;

if (!head) cout << "Список пуст!";

else

for (Node<T> \*node = head; node != nullptr; node = node->next)

{

cout << "-------------------------------" << endl;

cout << setw(3) << i + 1 << " " << node->data;

cout << endl;

i++;

}

cout << "-------------------------------" << endl;

cout << endl;

}

// геттер на пользователя

Account getUser(const string& username)

{

for (int i = 0; i < amount; i++)

{

if (this->operator[](i).getUsername() == username)

{

return this->operator[](i);

}

}

Account user(username, 0);

this->push\_tail(user);

string file = username + ".txt";

ofstream of(file);

of.close();

return user;

}

};

**File.h**

#pragma once

#include "List.h"

#include "Loan.h"

#include "Deposit.h"

#include "Account.h"

#include "MoneyTransfer.h"

#include <fstream>

template <class F>

class File

{

public:

File() {};

~File() {};

static void FileOpenAccount(List<F> &tmp, const char \*file);

static void FileSaveAccount(List<F> &tmp, const char \*file);

static void FileOpenDeposit(List<F> &tmp, const char \*file);

static void FileSaveDeposit(List<F> &tmp, const char \*file);

static void FileOpenLoan(List<F> &tmp, const char \*file);

static void FileSaveLoan(List<F> &tmp, const char \*file);

static void FileOpenMoneyTransfer(List<F> &tmp, const char \*file);

static void FileSaveMoneyTransfer(List<F> &tmp, const char \*file, string username);

};

template <class F>

void File<F>::FileOpenAccount(List <F> &tmp, const char \*file)

{

ifstream in;

in.open(file, ios::in);

if (!in.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

while (true)

{

string name;

double balance;

in >> name >> balance;

if (!in || name == "-1") break;

F obj(name, balance);

tmp.push\_tail(obj);

}

in.close();

}

template <class F>

void File<F>::FileSaveAccount(List <F> &tmp, const char \*file)

{

ofstream on;

on.open(file, ios\_base::out | ios\_base::trunc);

if (!on.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

for (int i = 0; i < tmp.size(); i++)

{

on << tmp[i].getUsername() << " " << tmp[i].getBalance() << endl;

}

on << "-1";

on.close();

}

template <class F>

void File<F>::FileOpenDeposit(List <F> &tmp, const char \*file)

{

ifstream in;

in.open(file, ios::in);

if (!in.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

while (true)

{

string name;

double balance;

int percent;

in >> name >> balance >> percent;

if (!in || name == "-1") break;

F obj(name, balance, percent);

tmp.push\_tail(obj);

}

in.close();

}

template <class F>

void File<F>::FileSaveDeposit(List <F> &tmp, const char \*file)

{

ofstream on;

on.open(file, ios\_base::out | ios\_base::trunc);

if (!on.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

for (int i = 0; i < tmp.size(); i++)

{

on << tmp[i].getUsername() << " " << tmp[i].getBalance() << " " << tmp[i].getPercent() << endl;

}

on << "-1";

on.close();

}

template <class F>

void File<F>::FileOpenLoan(List <F> &tmp, const char \*file)

{

ifstream in;

in.open(file, ios::in);

if (!in.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

while (true)

{

string name, partner;

double balance;

int percent;

in >> name >> balance >> percent >> partner;

if (!in || name == "-1") break;

F obj(name, balance, percent, partner);

tmp.push\_tail(obj);

}

in.close();

}

template <class F>

void File<F>::FileSaveLoan(List <F> &tmp, const char \*file)

{

ofstream on;

on.open(file, ios\_base::out | ios\_base::trunc);

if (!on.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

for (int i = 0; i < tmp.size(); i++)

{

on << tmp[i].getUsername() << " " << tmp[i].getBalance() << " " << tmp[i].getPercent() <<

" " << tmp[i].getPartnerUsername() << endl;

}

on << "-1";

on.close();

}

template <class F>

void File<F>::FileOpenMoneyTransfer(List <F> &tmp, const char \*file)

{

ifstream in;

in.open(file, ios::in);

if (!in.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

while (true)

{

string name;

double balance;

string partner;

in >> name >> balance >> partner;

if (!in || name == "-1") break;

F obj(name, balance, partner);

tmp.push\_tail(obj);

in.get();

}

in.close();

}

template <class F>

void File<F>::FileSaveMoneyTransfer(List <F> &tmp, const char \*file, string username)

{

ofstream on;

on.open(file, ios\_base::out | ios\_base::trunc);

if (!on.is\_open())

{

cout << "Невозможно открыть файл!" << endl;

return;

}

for (int i = 0; i < tmp.size(); i++)

{

on << username << " " << tmp[i].getBalance() << " " << tmp[i].getPartnerUsername() << endl;

}

on << "-1";

on.close();

}

**Interface.h**

#pragma once

#include <iostream>

#include <conio.h>

#include <fstream>

#include "List.h"

#include "File.h"

#include "Loan.h"

#include "Deposit.h"

#include "Account.h"

#include "Exception.h"

#include "ListException.h"

#include "InputException.h"

template<class T>

class Interface

{

public:

static void outputTransfer(T& MainUser);

static void showAccounts(List<Account>& list);

static void transferTo(T& MainUser, List<T>& list);

static void cancelTransfer(T& MainUser, List<T>& list);

static Deposit getDeposit(List<T>& list, Account& MainUser);

static void getLoans(List<T>& list, List<T>&loanList, Account& MainUser);

static void putDeposit(Deposit& MainDeposit, List<Deposit>& listD, T& MainUser, List<T>& list);

static void outDeposit(Deposit& MainDeposit, List<Deposit>& listD, T& MainUser, List<T>& list);

static void payLoan(Account& user, List<Account>& bankBookList, List<Loan> & loanList, List<Loan> & loans);

static void menu(Account& user, List<Deposit>& depositList, List<Loan> & loanList, List<Account>& bankBookList);

static void deleteAccount(Account& user, List<Account>& bankBookList, List<Deposit>& depositList, List<Loan> & loanList);

static void createLoan(Deposit& MainDeposit, Account& user, List<Account>& bankBookList, List<Loan> & loanList, List<Loan> & loans);

};

template<class T>

void Interface<T>::menu(Account& MainUser, List<Deposit>& deposit, List<Loan> & loanList, List<Account>& bankBookList)

{

Deposit MainDeposit = Interface<Deposit>::getDeposit(deposit, MainUser);

thread\* Thr = new thread(updateBalance, &MainDeposit);

Thr->detach();

List<Loan> loans;

Interface<Loan>::getLoans(loanList, loans, MainUser);

char input;

do

{

system("cls");

cout << "Выберите действие, которое хотите совершить" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 1. | Пополнить средства на депозите |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 2. | Вывести средства с депозита |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 3. | Посмотреть текущий баланс |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 4. | Перевести деньги на другой счёт |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 5. | Посмотреть счета всех аккаунтов |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 6. | Отменить действие с переводом средств |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 7. | Посмотреть информацию о займах |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 8. | Взять займ |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 9. | Покрыть займ |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| s. | Посмотреть действия с переводами средств |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| d. | Удалить аккаунт |" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| 0. | Выход из программы |" << endl;

cout << "---------------------------------------------------------" << endl;

switch ((input = \_getch()))

{

case '1': // 1. Пополнить средства на депозите

{

system("cls");

Interface<Account>::putDeposit(MainDeposit, deposit, MainUser, bankBookList);

system("pause");

break;

}

case '2': // 2. Вывести средства с депозита

{

system("cls");

Interface<Account>::outDeposit(MainDeposit, deposit, MainUser, bankBookList);

system("pause");

break;

}

case '3': // 3. Посмотреть текущий баланс

{

system("cls");

cout << "-------------------------------" << endl;

cout << "Имя пользователя: " << MainUser.getUsername() << endl;

cout << "-------------------------------" << endl;

cout << "Ваш баланс составляет: ";

cout << MainDeposit;

cout << "-------------------------------" << endl;

system("pause");

break;

}

case '4': // 4. Перевод денег на другой счёт

{

system("cls");

Interface<Account>::transferTo(MainDeposit, bankBookList);

system("pause");

break;

}

case '5': // 5. Посмотреть счета всех аккаунтов

{

system("cls");

Interface<Account>::showAccounts(bankBookList);

system("pause");

break;

}

case '6': // 6. Отмена последнего действия с переводом средств

{

system("cls");

Interface<Account>::cancelTransfer(MainUser, bankBookList);

system("pause");

break;

}

case '7': // 7. Просмотреть информацию о займах

{

system("cls");

if (loans.size() < 1) { cout << "Список пуст!" << endl; }

else { loans.output\_loans(); }

system("pause");

break;

}

case '8': // 8. Взять займ

{

system("cls");

Interface<Account>::createLoan(MainDeposit, MainUser, bankBookList, loanList, loans);

system("pause");

break;

}

case '9': // 9. Покрыть займ

{

system("cls");

if (loans.size() < 1)

{

cout << "Список пуст!" << endl;

system("pause");

break;

}

Interface<Account>::payLoan(MainUser, bankBookList, loanList, loans);

system("pause");

break;

}

case 's': // s. Посмотреть действия с переводом средств

{

system("cls");

Interface<Account>::outputTransfer(MainUser);

system("pause");

break;

}

case 'd': // d. Удалить аккаунт

{

system("cls");

cout << "Вы действительно желаете удалить аккаунт? " << endl;

cout << "1 - Да" << endl;

cout << "2 - Нет" << endl;

int deleteAccount = CheckInput(1, 2);

if (deleteAccount == 1)

{

Interface<Account>::deleteAccount(MainUser, bankBookList, deposit, loanList);

system("pause");

input = '0';

}

else

{

system("cls");

cout << "Отмена удаления аккаунта." << endl;

system("pause");

}

break;

}

case '0':

{

return;

}

default:

{

system("cls");

cout << "Введена неверная команда!" << endl;

cout << "Пожалуйста, повторите ввод." << endl;

system("pause");

break;

}

}

} while (input != '0');

}

template<class T>

void Interface<T>::putDeposit(Deposit& MainDeposit, List<Deposit>& listD, T& MainUser, List<T>& list)

{

system("cls");

cout << "Введите сумму для пополнения вашего депозита: ";

double balance = CheckInput(0, 1000000);

MainUser.setBalance(MainUser.getBalance() + balance);

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

list[i].setBalance(MainUser.getBalance());

}

}

MainDeposit.setBalance(MainDeposit.getBalance() + balance);

for (int i = 0; i < listD.size(); i++)

{

if (listD[i].getUsername() == MainUser.getUsername())

{

listD[i].setBalance(MainDeposit.getBalance());

}

}

system("pause");

}

template<class T>

void Interface<T>::outDeposit(Deposit& MainDeposit, List<Deposit>& listD, T& MainUser, List<T>& list)

{

system("cls");

cout << MainDeposit;

cout << "Введите сумму для снятия средств с вашего депозита: ";

double balance = CheckInput(0, MainDeposit.getBalance());

MainDeposit.setBalance(MainDeposit.getBalance() - balance);

for (int i = 0; i < listD.size(); i++)

{

if (listD[i].getUsername() == MainUser.getUsername())

{

listD[i].setBalance(MainDeposit.getBalance());

}

}

MainUser.setBalance(MainUser.getBalance() - balance);

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

list[i].setBalance(MainUser.getBalance());

}

}

system("pause");

}

template<class T>

void Interface<T>::transferTo(T& MainUser, List<T>& list)

{

system("cls");

list.output();

int num = CheckInput(1, list.size());

MoneyTransfer moneyTr;

while (true)

{

cin >> moneyTr;

if (moneyTr.getBalance() > MainUser.getBalance())

{

cout << "Недостаточно средств!" << endl;

continue;

}

else

{

list[num - 1].setBalance(list[num - 1].getBalance() + moneyTr.getBalance());

MainUser.setBalance(MainUser.getBalance() - moneyTr.getBalance());

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

list[i].setBalance(MainUser.getBalance());

}

}

moneyTr.setPartnerUsername(list[num - 1].getUsername());

moneyTr.setUsername(MainUser.getUsername());

string text = MainUser.getUsername() + ".txt";

const char\* file = text.c\_str();

List<MoneyTransfer> listMoney;

File<MoneyTransfer>::FileOpenMoneyTransfer(listMoney, file);

listMoney.push\_tail(moneyTr);

File<MoneyTransfer>::FileSaveMoneyTransfer(listMoney, file, MainUser.getUsername());

return;

}

break;

}

}

template<class T>

void Interface<T>::showAccounts(List<Account>& list)

{

list.output();

}

template<class T>

void Interface<T>::cancelTransfer(T& MainUser, List<T>& list)

{

system("cls");

string text = MainUser.getUsername() + ".txt";

const char\* file = text.c\_str();

List<MoneyTransfer> listMoney = List<MoneyTransfer>();

File<MoneyTransfer>::FileOpenMoneyTransfer(listMoney, file);

if (listMoney.size() == 0)

{

cout << "Ошибка 2.1:" << endl;

cout << "Список пуст!" << endl;

return;

}

MoneyTransfer last = listMoney[list.size() - 1];

listMoney.pop\_tail();

MainUser.setBalance(MainUser.getBalance() + last.getBalance());

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

list[i].setBalance(MainUser.getBalance());

}

if (list[i].getUsername() == last.getPartnerUsername())

{

list[i].setBalance(list[i].getBalance() - last.getBalance());

}

}

File<MoneyTransfer>::FileSaveMoneyTransfer(listMoney, file, MainUser.getUsername());

}

template<class T>

Deposit Interface<T>::getDeposit(List<T>& list, Account& MainUser)

{

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

return list[i];

}

}

Deposit mainDeposit;

mainDeposit.setUsername(MainUser.getUsername());

list.push\_tail(mainDeposit);

return mainDeposit;

}

template<class T>

void Interface<T>::getLoans(List<T>& list, List<T>& loanList, Account& MainUser)

{

for (int i = 0; i < list.size(); i++)

{

if (list[i].getUsername() == MainUser.getUsername())

{

loanList.push\_tail(list[i]);

thread\* Thr = new thread(updateBalance, &loanList[i]);

Thr->detach();

}

}

}

template<class T>

void Interface<T>::outputTransfer(T& MainUser)

{

system("cls");

string text = MainUser.getUsername() + ".txt";

const char\* file = text.c\_str();

List<MoneyTransfer> listMoney;

File<MoneyTransfer>::FileOpenMoneyTransfer(listMoney, file);

if (!listMoney.size())

{

cout << "Ошибка 2.1:" << endl;

cout << "Список пуст!" << endl;

return;

}

else

{

listMoney.output\_loans();

}

}

template<class T>

void Interface<T>::deleteAccount(Account& user, List<Account>& bankBookList, List<Deposit>& deposit, List<Loan> & loanList)

{

for (int i = 0; i < bankBookList.size(); i++)

{

if (bankBookList[i].getUsername() == user.getUsername())

{

if (i == 0) bankBookList.pop\_head();

else if (i == bankBookList.size() - 1) bankBookList.pop\_tail();

else bankBookList.pop(i);

i--;

}

}

for (int i = 0; i < deposit.size(); i++)

{

if (deposit[i].getUsername() == user.getUsername())

{

if (i == 0) deposit.pop\_head();

else if (i == deposit.size() - 1) deposit.pop\_tail();

else deposit.pop(i);

i--;

}

}

for (int i = 0; i < loanList.size(); i++)

{

if (loanList[i].getUsername() == user.getUsername())

{

for (int j = 0; j < bankBookList.size(); j++)

{

if (bankBookList[i].getUsername() == loanList[i].getPartnerUsername())

{

bankBookList[i].setBalance(bankBookList[i].getBalance() + loanList[i].getBalance());

}

}

if (i == 0) loanList.pop\_head();

else if (i == loanList.size() - 1) loanList.pop\_tail();

else loanList.pop(i);

i--;

}

}

}

template<class T>

void Interface<T>::createLoan(Deposit& MainDeposit, Account& user, List<Account>& bankBookList, List<Loan> & loanList, List<Loan> & loans)

{

Loan newLoan;

cin >> newLoan;

loanList.push\_tail(newLoan);

thread\* Thr = new thread(updateBalance, &loanList[loanList.size() - 1]);

Thr->detach();

loans.push\_tail(newLoan);

thread\* Thr2 = new thread(updateBalance, &loans[loans.size() - 1]);

Thr2->detach();

for (int i = 0; i < bankBookList.size(); i++)

{

if (bankBookList[i].getUsername() == newLoan.getPartnerUsername())

{

bankBookList[i].setBalance(bankBookList[i].getBalance() - newLoan.getBalance());

user.setBalance(user.getBalance() + newLoan.getBalance());

}

if (bankBookList[i].getUsername() == user.getUsername())

{

bankBookList[i].setBalance(bankBookList[i].getBalance() + newLoan.getBalance());

MainDeposit.setBalance(MainDeposit.getBalance() + newLoan.getBalance());

}

}

}

template<class T>

void Interface<T>::payLoan(Account& user, List<Account>& bankBookList, List<Loan> & loanList, List<Loan> & loans)

{

if (loans.size() < 1)

{

throw ListException(1);

}

loans.output();

cout << "Выберите займ для покрытия: ";

int num = CheckInput(1, loans.size());

num--;

cout << "Введите сумму для покрытия займа: ";

double balance = CheckInput(0, user.getBalance());

for (int i = 0; i < bankBookList.size(); i++)

{

if (bankBookList[i].getUsername() == loans[num].getPartnerUsername())

{

if (balance < loans[num].getBalance())

{

bankBookList[i].setBalance(bankBookList[i].getBalance() + balance);

user.setBalance(user.getBalance() - balance);

for (int j = 0; j < bankBookList.size(); j++)

{

if (bankBookList[j].getUsername() == user.getUsername())

{

bankBookList[j].setBalance(bankBookList[j].getBalance() - balance);

}

}

loans[num].setBalance(loans[num].getBalance() - balance);

}

else if (balance == loans[num].getBalance())

{

bankBookList[i].setBalance(bankBookList[i].getBalance() + balance);

user.setBalance(user.getBalance() - balance);

for (int j = 0; j < bankBookList.size(); j++)

{

if (bankBookList[j].getUsername() == user.getUsername())

{

bankBookList[j].setBalance(bankBookList[j].getBalance() - balance);

}

}

loans[num].setBalance(0);

}

else

{

bankBookList[i].setBalance(bankBookList[i].getBalance() + loans[num].getBalance());

user.setBalance(user.getBalance() - loans[num].getBalance());

for (int j = 0; j < bankBookList.size(); j++)

{

if (bankBookList[j].getUsername() == user.getUsername())

{

bankBookList[j].setBalance(bankBookList[j].getBalance() - loans[num].getBalance());

}

}

loans[num].setBalance(0);

}

}

}

for (int i = 0; i < loanList.size(); i++)

{

if (loanList[i].getBalance() == 0)

loanList.pop(i);

}

if (loans[num].getBalance() == 0)

{

loans.pop(num);

}

}