BalanceChange.h

```
#pragma once
#include <iostream>
using namespace std;
class BalanceChange
{
protected:
		double sum;
public:
		BalanceChange() { sum = 0; }
		~BalanceChange() { }

		double getSum() { return sum; }
		void setSum(double _sum) { this->sum = _sum; }
};

int InputCheck(double inter1, double inter2);
```

Consum.h

```
#pragma once
#include "BalanceChange.h"

class Consum : public BalanceChange
{
		int type;
		bool debt;
public:
		Consum() { type = 0, debt = false; }
		Consum(const Consum &tmp);
		Consum(int _type, double _sum, bool _debt);
		~Consum() {}

		double getSum() { return this->sum; }
		int getType() { return this->type; }
		bool getDebt() { return this->debt; }

		void setSum(double _sum) { sum = _sum; }
		void setType(int _type) { type = _type; }
		void setDebt(int _debt) { debt = _debt; }

		friend void operator >> (istream & is, Consum & tmp);
		friend void operator << (ostream & os, Consum & tmp);
		friend bool operator== (Consum tmp1, Consum tmp2);
		void operator = (const Consum & tmp);
};
```

Exception.h

```
#pragma once
#include <iostream>
class Exception
{
protected:
		int error;
public:
		Exception()
		{
			error = 0;
		}
		Exception(int n)
		{
			error = n;
```

```
            }
};
```

File.h

```
#pragma once
#include "List.h"

template <class X>
class File
{
public:
            File() {};
            ~File() {};
            void FileSaveIncome (List<X> &tmp, const char *file);
            void FileSaveConsum(List<X> &tmp, const char *file);
            void FileOpenIncome(List<X> &tmp, const char *file);
            void FileOpenConsum(List<X> &tmp, const char *file);
};

template <class X>
void File<X>::FileOpenIncome(List <X> &tmp, const char *file)
{
            ifstream in;
            in.open(file, ios::in);
            if (!in.is_open())
            {
                        cout << "Невозможно открыть файл!" << endl;
                        return;
            }
            while (true)
            {
                                    float tmp1;
                                    int tmp2;
                                    in >> tmp1 >> tmp2;
                                    if (tmp1 == -1) break;
                                    Income obj(tmp2, tmp1);
                                    tmp.push_tail(obj);
            }
            in.close();
}

template <class X>
void File<X>::FileOpenConsum(List <X> &tmp, const char *file)
{
            ifstream in;
            in.open(file, ios::in);
            if (!in.is_open())
            {
                        cout << "Невозможно открыть файл!" << endl;
                        return;
            }
            while (true)
            {
                        float tmp1;
                        int tmp2;
                        bool tmp3;
                        in >> tmp1 >> tmp2 >> tmp3;
                        if (tmp1 == -1) break;
                        Consum obj(tmp2, tmp1, tmp3);
                        tmp.push_tail(obj);
            }
            in.close();
}
```

```
template <class X>
void File<X>::FileSaveIncome(List <X> &tmp, const char *file)
{
		ofstream on;
		on.open(file, ios::in | ios_base::trunc);
		if (!on.is_open())
		{
			cout << "Невозможно открыть файл!" << endl;
			return;
		}
		for (int i = 0; i<tmp.size(); i++)
		{
			on << tmp[i].getSum() << " " << tmp[i].getType() << endl;
		}
		on << "-1";
		on.close();
}

template <class X>
void File<X>::FileSaveConsum(List <X> &tmp, const char *file)
{
		ofstream on;
		on.open(file, ios::in | ios_base::trunc);
		if (!on.is_open())
		{
			cout << "Невозможно открыть файл!" << endl;
			return;
		}
		for (int i = 0; i<tmp.size(); i++)
		{
			on << tmp[i].getSum() << " " << tmp[i].getType() << " " <<
tmp[i].getDebt() << endl;
		}
		on << "-1";
		on.close();
}
```

Income.h

```
#pragma once
#include "BalanceChange.h"

class Income : public BalanceChange
{
		int type;
public:
		Income() { type = 0; }
		Income(const Income &tmp);
		Income(int _type, double _sum);
		~Income() {}

		double getSum() { return this->sum; }
		int getType() { return this->type; }

		void setSum(double _sum) { sum = _sum; }
		void setType(int _type) { type = _type; }

		friend void operator >> (std::istream & is, Income & tmp);
		friend void operator << (std::ostream & os, Income & tmp);
		friend bool  operator== (Income tmp1, Income tmp2);
		void operator = (const Income & tmp);
};
```

InputException.h

```
#pragma once

#include "exception.h"

class InputException :public Exception
{
public:
		InputException() :Exception() {};
		InputException(int n) :Exception(n) {};
		void NumError();
};
```

List.h

```
#pragma once
#include <iostream>
#include <iomanip>
#include <string>
#include "Income.h"
#include "Consum.h"
using namespace std;

template<typename T>
struct Node
{
		T data;
		Node<T> *next;
		Node<T> *prev;
};

template<typename T>
class List
{
protected:
		template<typename T> friend class Iterator;
		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;
}
//Доавление 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;
		}
		Node<T> *node = new Node<T>;
		node->data = input_object;
		node->next = nullptr;
		node->prev = tail;
		tail->next = node;
		tail = node;
		amount++;
		return;
}

//Удаление от 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;
		tail = tail->prev;
		if (tail)
				tail->next = nullptr;
		delete node;
```

```
                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;
        }

        // удаление элемента из списка
        void delete_element(Node<T> *node)
        {
                if (head == node) head = head->next;
                if (tail == node) tail = tail->prev;
                if (node->next != nullptr) node->next->prev = node->prev;
                if (node->prev != nullptr) node->prev->next = node->next;
                delete node;
                node = nullptr;
                this->amount--;
        }

        double getBalance(List<Income> &income, List<Consum> &consum)
        {
                double tempBalance=0;
                if (income.size()>0)
                for (Iterator<Income> i = income.begin(); i != Iterator
<Income>(income.end()); i++)
                {
                        tempBalance += (*i).getSum();
                }
                if(consum.size()>0)
                for (Iterator<Consum> j = consum.begin(); j !=
Iterator<Consum>(consum.end()); j++)
                {
                        tempBalance -= (*j).getSum();
                }
                return tempBalance;
        }

        // 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 List<T>::output()
        {
                int i = 0;
                if (!head) cout << "Список пуст!";
                else
                        for (Node<T> *node = head; node != nullptr; node
= node->next)
                        {
                                cout << setw(3) << i+1 << node-
>data;
```

```
                                                cout << endl;
                                                i++;
                                        }
                        }
                        //Вывод по типу
                        void List<T>::output_for_type(int _type)
                        {
                                int i = 0;
                                if (!head) cout << "Список пуст!";
                                else
                                        for (Node<T> *node = head; node != nullptr; node
= node->next)
                                        {
                                                if (node->data.getType() == _type)
                                                {
                                                        cout << setw(3) << i +
1 << node->data;
                                                        cout << endl;
                                                        i++;
                                                }
                                        }
                        }
};

template <typename T>
class Iterator
{
                Node<T> *current;
public:
                Iterator()
                {
                        current = nullptr;
                }
                Iterator(List<T> &container)
                {
                        current = container.head;
                }
                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;
		}

		bool operator==(Iterator &iter)
		{
			if (current == nullptr && iter.current == nullptr)
				return true;
			if (current == nullptr || iter.current == nullptr)
				return false;
			if (current->data == iter.current->data && current->next ==
iter.current->next && current->prev == iter.current->prev)
				return true;
			return false;
		}
		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;
			if (current->data == iter.current->data && current->next ==
iter.current->next && current->prev == iter.current->prev)
				return false;
			return true;
		}
};
```

ListException.h

```
#pragma once

#include "exception.h"

class ListException :public Exception
{
public:
		ListException() : Exception() {};
		ListException(int n) : Exception(n) {};
		void NumError();
};
```

User.h

```
#pragma once
#include <string>
using namespace std;

class User
{
		string username;
		double balance;
public:
		User() { balance = 0; };
		User(char* currentName, double currentMoney);
		User(const User &tmp);
		~User() {};
```

```
            string getUsername() const { return this->username; }
            double getBalance() { return this->balance; }

            void setUsername(string tmp) { this->username = tmp; }
            void setBalance(double _balance) {this->balance = _balance; }

            string InputCharCheck(string input);
            bool CheckCorrect(string input, bool fl);

            friend void operator >> (std::istream & is, User & tmp);
};
```

BalanceChange.cpp

```
#include "BalanceChange.h"
#include "InputException.h"

int InputCheck(double inter1, double inter2)
{
            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<inter1 || tmp>inter2)
                                    {
                                                flag = false;
                                                throw 5;
                                    }
                        }
                        catch (InputException e)
                        {
                                    e.NumError();
                                    cin.sync();
                                    cin.clear();
                                    rewind(stdin);
                        }
                        catch (...)
                        {
                                    cin.sync();
                                    cin.clear();
                                    rewind(stdin);
                                    cout << "Введите число в интервале от " << inter1
<< " до " << inter2 << endl;
                        }
            } while (!flag);
            return tmp;
}
```

Consum.cpp

```
#include "Consum.h"
```

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include "InputException.h"

using namespace std;

Consum::Consum(const Consum &tmp)
{
		sum = tmp.sum;
		debt = tmp.debt;
		type = tmp.type;
}
Consum::Consum(int _type, double _sum, bool _debt)
{
		type = _type;
		sum = _sum;
		debt = _debt;
}

void operator >> (std::istream & is, Consum & tmp)
{
		cout << "Введите сумму расхода: ";
		tmp.sum=InputCheck(0, 999999);
		cout << "Выберите тип расхода:" << endl;
		cout << "1. Еда" << endl;
		cout << "2. Одежда" << endl;
		cout << "3. Медицина и здоровье" << endl;
		cout << "4. Автомобиль" << endl;
		cout << "5. Дом" << endl;
		cout << "6. Интернет и мобильная связь" << endl;
		cout << "7. Транспорт" << endl;
		cout << "8. Другое" << endl;
		tmp.type = InputCheck(1, 8);
		if (tmp.type == 8)
		{
					cout << "Это долг?" << endl;
					cout << "1 - Да" << endl;
					cout << "Любой другой символ - нет" << endl;
					switch (_getch())
					{
					case '1':
					{
							tmp.debt = true; break;
					}
					default:
					{
							tmp.debt = false; break;
					}
					}
		}
}
void operator << (std::ostream & os, Consum & tmp)
{
		os << setw(12) << tmp.sum;
		switch (tmp.type)
		{
		case 1:
		{
				os << setw(25) << "Еда";
				break;
		}
		case 2:
		{
				os << setw(25) << "Одежда";
				break;
```

```
		}
		case 3:
		{
				os << setw(25) << "Медицина и здоровье";
				break;
		}
		case 4:
		{
				os << setw(25) << "Автомобиль";
				break;
		}
		case 5:
		{
				os << setw(25) << "Домашние расходы";
				break;
		}
		case 6:
		{
				os << setw(25) << "Интернет и моб. связь";
				break;
		}
		case 7:
		{
				os << setw(25) << "Транспорт";
				break;
		}
		case 8:
		{
				os << setw(25) << "Другое";
				if (tmp.debt)
						os << setw(8) << "Да";
				break;
		}
		}
}
void Consum::operator = (const Consum & tmp)
{
		sum = tmp.sum;
		debt = tmp.debt;
		type = tmp.type;
}

bool  operator== (Consum tmp1, Consum tmp2)
{
		if (tmp1.debt == tmp2.debt && tmp1.sum == tmp2.sum && tmp1.type ==
tmp2.type)
				return true;
		return false;
}
```

Income.cpp

```
#include "Income.h"
#include <iostream>
#include <iomanip>
#include <conio.h>
#include "InputException.h"

using namespace std;

Income::Income(const Income &tmp)
{
		sum = tmp.sum;
		type = tmp.type;
}
```

```
Income::Income(int _type, double _sum)
{
		type = _type;
		sum = _sum;
}

void operator >> (std::istream & is, Income & tmp)
{
		cout << "Введите сумму дохода: ";
		tmp.sum=InputCheck(0,999999);
		cout << "Выберите тип дохода:" << endl;
		cout << "1. Зарплата" << endl;
		cout << "2. Подарок" << endl;
		cout << "3. Дивиденды" << endl;
		cout << "4. Другое" << endl;
		tmp.type = InputCheck(1, 4);
}
void operator << (std::ostream & os, Income & tmp)
{
		os << setw(12) << tmp.sum;
		switch (tmp.type)
		{
		case 1:
		{
				os << setw(15) << "Зарплата";
				break;
		}
		case 2:
		{
				os << setw(15) << "Подарок";
				break;
		}
		case 3:
		{
				os << setw(15) << "Дивиденды";
				break;
		}
		case 4:
		{
				os << setw(15) << "Другое";
				break;
		}
		}
}
void Income::operator = (const Income & tmp)
{
		sum = tmp.sum;
		type = tmp.type;
}

bool  operator== (Income tmp1, Income tmp2)
{
		if (tmp1.sum == tmp2.sum && tmp1.type == tmp2.type)
				return true;
		return false;
}
```

InputException.cpp

```
#include "InputException.h"
#include <iostream>

using namespace std;

void InputException::NumError()
```

```
{
		switch (error)
		{
		case 1:
				cout << "Ошибка 1.1" << endl;
				cout << "Вы ввели недопустимый символ!" << endl;
				cout << "Разрешён ввод букв латинского алфавита и цифр." <<
endl << endl;
				break;
		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 3.1" << endl;
				cout << "Вы ввели символы вместо числа либо превышен предел по
вводу чисел!" << endl << endl;
				break;
		default:
				puts("Ошибка ввода");
				puts("Повторите ввод");
				break;
		}
}
```

ListException.cpp

```
#include "ListException.h"
#include <iostream>

using namespace std;

void ListException::NumError()
{
		switch (error)
		{
		case 1:
				cout << "Ошибка 2.1" << endl;
				cout << "Список пуст" << endl;
				break;
		}
}
```

Main.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include <conio.h>
#include <fstream>
#include "BalanceChange.h"
#include "User.h"
#include "List.h"
#include "Consum.h"
#include "Income.h"
#include <windows.h>
```

```
#include <vector>
#include "File.h"
using namespace std;

int main()
{
		setlocale(LC_ALL, "Russian");
		ifstream if_user("user.txt");
		User MainUser;
		cout << "Введите имя пользователя: ";
		cin >> MainUser;
		ofstream of_user("user.txt");
		of_user << MainUser.getUsername() << " " << MainUser.getBalance();
		of_user.close();

		HWND hWnd = GetForegroundWindow();	//запрет аварийного закрытия окна
(через крестик)
		EnableWindow(hWnd, false);

		List<Consum> consum;
		List<Income> income;

		File<Income> fi;
		File<Consum> fc;

		fi.FileOpenIncome(income, "income.txt");
		fc.FileOpenConsum(consum, "consum.txt");

		vector<Consum> CCancel;
		vector<int> c;
		vector<Income> ICancel;
		vector<int> i;

		while (true)
		{
			MainUser.setBalance(income.getBalance(income, consum));
			system("cls");
			cout << "Вас приветствует система Домашняя бухгалтерия!" <<
endl;
			cout << "Добро пожаловать, " << MainUser.getUsername() << "!"
<< endl;
			cout << "Ваш текущий баланс составляет " <<
MainUser.getBalance() <<  "р." << endl << endl;
			cout << "Выберите действие, которое хотите совершить:" << endl;
			cout << "1. Добавить доход в базу" << endl;
			cout << "2. Добавить расход в базу" << endl;
			cout << "3. Посмотреть список доходов" << endl;
			cout << "4. Посмотреть список расходов" << endl;
			cout << "5. Удалить последний доход из базы" << endl;
			cout << "6. Удалить последний расход из базы" << endl;
			cout << "7. Изменить доход" << endl;
			cout << "8. Изменить расход" << endl;
			cout << "9. Вывод доходов по типу" << endl;
			cout << "q. Вывод расходов по типу" << endl;
			cout << "c. Отмена предыдущего действия с доходом" << endl;
			cout << "v. Отмена предыдущего действия с расходом" << endl;
			cout << "r. Обновить состояние" << endl;
			cout << "0. Выход из программы" << endl;
			switch (_getch())
			{
			case '1':		//Добавить доход
			{
				system("cls");
				Income tmp;
				cin >> tmp;
				income.push_tail(tmp);
```

```
                                        ICancel.insert(ICancel.end(), tmp);
                                        i.insert(i.end(), 1);
                                        system("pause");
                                        break;
                                }
                                case '2':               //Добавить расход
                                {
                                        system("cls");
                                        Consum tmp;
                                        cin >> tmp;
                                        consum.push_tail(tmp);
                                        CCancel.insert(CCancel.end(), tmp);
                                        c.insert(c.end(), 1);
                                        system("pause");
                                        break;
                                }
                                case '3':               //Посмотреть доходы
                                {
                                        system("cls");
                                        if (income.size() > 0)
                                        {
                                                cout << setw(3) << "#" << setw(12)
<< "Сумма" << setw(15) << "Тип" << endl << endl;
                                                income.output();
                                                cout << endl;
                                        }
                                        else cout << "Список пуст!" << endl;
                                        system("pause");
                                        break;
                                }
                                case '4':               //Посмотреть расходы
                                {
                                        system("cls");
                                        if (consum.size() > 0)
                                        {
                                                cout << setw(3) << "#" << setw(12)
<< "Сумма" << setw(25) << "Тип" << setw(8) << "Долг" << endl << endl;
                                                consum.output();
                                                cout << endl;
                                        }
                                        else cout << "Список пуст!" << endl;
                                        system("pause");
                                        break;
                                }
                                case '5':               //Удалить доход
                                {
                                        system("cls");
                                        if (income.size() > 0)
                                        {
                                                Income tmp = income[income.size()];
                                                ICancel.insert(ICancel.end(), tmp);
                                                i.insert(i.end(), 2);
                                                income.pop_tail();
                                        }

                                        system("pause");
                                        break;
                                }
                                case '6':               //Удалить расход
                                {
                                        system("cls");
                                        if (consum.size() > 0)
                                        {
                                                Consum tmp = consum[consum.size()];
                                                CCancel.insert(CCancel.end(), tmp);
                                                c.insert(c.end(), 2);
```

```
                        consum.pop_tail();
                }
                system("pause");
                break;
        }
        case '7':           //Изменить доход
        {
                system("cls");
                cout << "Введите номер изменяемого дохода: ";
                int tempNumber=InputCheck(1, consum.size());
                Income tmp;
                cin >> tmp;
                income[tempNumber-1] = tmp;
                system("pause");
                break;
        }
        case '8':           //Изменить расход
        {
                system("cls");
                cout << "Введите номер изменяемого расхода: ";
                int tempNumber=InputCheck(1,consum.size());
                Consum tmp;
                cin >> tmp;
                consum[tempNumber-1] = tmp;
                system("pause");
                break;
        }
        case '9':           //Просмотр по типу дохода
        {
                system("cls");
                cout << "Выберите тип дохода: " << endl;
                cout << "1. Зарплата" << endl;
                cout << "2. Подарок" << endl;
                cout << "3. Дивиденды" << endl;
                cout << "4. Другое" << endl;
                int tempType=InputCheck(1,4);
                system("cls");
                income.output_for_type(tempType);
                system("pause");
                break;
        }
        case 'q':           //Просмотр по типу расхода
        {
                system("cls");
                cout << "Выберите тип расхода: " << endl;
                cout << "1. Еда" << endl;
                cout << "2. Одежда" << endl;
                cout << "3. Медицина и здоровье" << endl;
                cout << "4. Автомобиль" << endl;
                cout << "5. Дом" << endl;
                cout << "6. Интернет и мобильная связь" << endl;
                cout << "7. Транспорт" << endl;
                cout << "8. Другое" << endl;
                int tempType = InputCheck(1, 8);
                system("cls");
                consum.output_for_type(tempType);
                system("pause");
                break;
        }
        case 'c':           //Отмена действия с доходом
        {
                if (i.back() == 1)
                {
                        income.pop_tail();
                        i.erase(i.end() - 1);
                        ICancel.erase(ICancel.end() - 1);
```

```
                                        }
                                        if (i.back() == 2)
                                        {
                                                income.push_tail(ICancel.back());
                                                i.erase(i.end() - 1);
                                                ICancel.erase(ICancel.end() - 1);
                                        }
                                        break;
                                }
                                case 'v':           //Отмена действия с расходом
                                {
                                        if (c.back() == 1)
                                        {
                                                consum.pop_tail();
                                                c.erase(c.end() - 1);
                                                CCancel.erase(CCancel.end() - 1);
                                        }
                                        if (c.back() == 2)
                                        {
                                                consum.push_tail(CCancel.back());
                                                c.erase(c.end() - 1);
                                                CCancel.erase(CCancel.end() - 1);
                                        }
                                        break;
                                }
                                case 'r':           //Обновление баланса (мой маленький
рудементик курсача)
                                {

       MainUser.setBalance(income.getBalance(income,consum));
                                        break;
                                }
                                case '0':           //Безопасный выход из программы
                                {
                                        fi.FileSaveIncome(income, "income.txt");
                                        fc.FileSaveConsum(consum, "consum.txt");
                                        exit (0);
                                }
                                }
                }
}
```

User.cpp

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include "User.h"
#include <string>
#include <iostream>
#include "InputException.h"

using namespace std;

User :: User(char* currentName, double currentMoney)
{

                username = currentName;
                balance = currentMoney;
}

User::User(const User &tmp)
{
                this->balance = tmp.balance;
                username = tmp.username;
}
```

```
bool User::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 User::InputCharCheck(string input)
{
		bool fl = true;
		do {
				try {
						User tmp;
						fflush(stdin);
						cin.sync();
						cin.clear();
						cin >> input;
						tmp.CheckCorrect(input, fl);
				}
				catch (InputException e)
				{

						e.NumError();
						cin.clear();
						cin.sync();
						InputCharCheck(input);
				}
				fflush(stdin);
		} while (!fl);
		return input;
}

void operator >> (std::istream & is, User &tmp)
{
		tmp.username = tmp.InputCharCheck(tmp.username);
}
```