Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ПетрГУ)

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

**БЮРО РИТУАЛЬНЫХ УСЛУГ**

Командное задание по курсу «Технология программирования»

Выполнили:

студенты 3 курса

физико-технического института, гр. 21316

Волкова Я. А.,

Егорова В. Л.

Проверил:

доц., канд. ф.-м. наук

Бульба А.В.

Петрозаводск 2020

**Цель работы**

Смоделировать ситуацию выполнения заказа на разработку программного обеспечения по сюжету «бюро ритуальных услуг», создать диаграммы, описывающие работу ПО.

**Программная реализация**

*Среда разработки:* IDE Visual Studio 2013 Express

*Язык:* С++

*Заголовочные файлы:*

AnnualReport.h

Содержит объявление класса AnnualReport.

Client.h

Содержит объявление класса Client.

ClientList.h

Содержит объявление класса ClientList.

ClientInputScreen.h

Содержит объявление класса ClientInputScreen.

Expense.h

Содержит объявление класса Expense.

ExpenseRecord.h

Содержит объявление класса ExpenseRecord.

ExpenseInputScreen.h

Содержит объявление класса ExpenseInputScreen.

Header.h

Содержит объявление глобальных методов, подключает к себе все остальные заголовочные файлы.

PayRow.h

Содержит объявление класса PayRow.

PayRecord.h

Содержит объявление класса PayRecord.

SumInputScreen.h

Содержит объявление класса SumInputScreen.

UserInterface.h

Содержит объявление класса UserInterface.

*Единицы компиляции:*

AnnualReport.cpp

Содержит тела методов класса AnnualReport: конструктор, display() – выводит годовой отчет.

Client.cpp

Содержит тела методов класса Client: конструктор, деструктор, геттеры getName(), getPhoneNumber().

ClientList.cpp

Содержит тела методов класса ClientList: деструктор, display() – выводит список клиентов с номерами телефонов, insertClient(Client\* ptrT) – записывает клиента в список клиентов, getphNum(string tName) – получает номер телефона по имени клиента.

ClientInputScreen.cpp

Содержит тела методов класса ClientInputScreen: setClietn() – запрашивает данные о клиенте и передает их в список клиентов.

ExpenseRecord.cpp

Содержит тела методов класса ExpenseRecord: деструктор, display(), который выводит все записанные расходы, displaySummary() – возвращает общую сумму расходов, insertExp(Expense\* ptrExp) – вставляет данные о расходах в конец таблицы.

ExpenseInputScreen.cpp

Содержит тела методов класса ExpenseInputScreen: конструктор, setExpense(), который записывает данные в таблицу расходов.

Source.cpp

Содержит тела глобальных методов getaLine(string& inStr), getaChar(), checkDigit(int& x).

Main.cpp

Содержит функцию main, которая запускает UserInterface. Подключает все необходимые заголовочные файлы.

PayRow.cpp

Содержит тела методов класса PayRow: конструктор, setPay(string cl, string ph, int m, float am, int serv), геттеры

PayRecord.cpp

Содержит тела методов класса PayRecord: деструктор, insertPay(string name, string phNo, int month, float amount, int serv) – создает строку с данными о сделке в объекте класса PayRow, display() – вывод таблицы оплаченных товаров/услуг, getSumOfPays() – возвращает общую сумму всех сделок.

SumInputScreen.cpp

Содержит тело метода setPay() класса SumInputScreen. setPay() осуществляет ввод данных об оплаченной услуге/товаре в таблицу.

UserInterface.cpp

Содержит тела методов класса UserInterface: конструктор, деструктор, метод interact(), осуществляющий взаимодействие остальных единиц компиляции.

**Пошаговое описание процесса разработки**

1. **Краткое словесное описание сюжета**

Наша команда экспертов занимается разработкой программного обеспечения на языке С++. В нашу фирму обратился заказчик по имени Светлов Александр Юрьевич, являющийся владельцем одного из бюро ритуальных услуг города Иваново. Он давно искал способ упростить свой нелегкий труд и избавиться от лишней бумажной волокиты. Александр Юрьевич решил не отставать от других предприятий, во всю внедряющих в свой рабочий процесс новейшие информационные технологии. И поэтому обратился к нам с задачей создания такой программы, которая помогла бы оптимизировать процесс добавления новых клиентов, записи проданных товаров и услуг, внесения данных по расходам в таблицу, отражающую общую прибыль и расходы за прошедший год.

Самым оптимальным решением, на наш взгляд, было бы спроектировать несколько таблиц, которые будут хранить вносимые заказчиком данные о расходах и доходах бюро ритуальных услуг. При этом интерфейс программы должен быть максимально понятен ее пользователю, а так же был весьма эргономичным.

Так же Александр Юрьевич выразил желание получить возможность такой функции в программе, которая могла бы выдавать годовой отчет с рассчитанной чистой прибылью.

Мы приняли все запросы и пожелания заказчика и приступили к разработке соответствующего программного обеспечения.

1. **Описание имеющихся у заказчика материалов (данных)**

Данные, имеющиеся у заказчика:

* данные о клиентах (имя клиента и его телефонный номер);
* данные о приобретенных клиентами товарах и услугах (имя клиента, тип товара/услуги, месяц осуществления сделки, сумма сделки);
* данные о расходах бюро (день и месяц оплаты, категория расходов, имя получателя оплаты, выплаченная сумма).

1. **Действующие субъекты и их функциональные обязанности**

В данном сюжете присутствует только один субъект (actor) – это работник бюро ритуальных услуг.

Единственный пользователь системы. Работник бюро может добавить нового клиента, ввести проданную услугу/товар, внести новые данные в таблицу расходов. Также, данный пользователь может посмотреть список всех проданных услуг и товаров, список расходов и годовой отчет.

1. **Список вариантов использования. Диаграмма вариантов использования**

* Запустить программу;
* Добавить клиента;
* Ввести проданную услугу/товар;
* Ввести расходы;
* Вывести список проданных товаров/услуг;
* Вывести таблицу расходов;
* Вывести годовой отчет.

Диаграмма вариантов использования представлена в Приложении, рис. 1.

1. **Описание вариантов использования**

*Запустить программу*

Когда запускается программа, на экран должно выводиться меню, из которого пользователь может выбрать нужное действие. Это может называться экраном пользовательского интерфейса.

*Добавить клиента*

На экране должно отобразиться сообщение, в котором программа просит пользователя ввести имя клиента и его номер телефона. Эта информация должна заноситься в таблицу.

*Ввести проданную услугу/товар*

Экран ввода услуги/товара запрашивает имя клиента. Если имя клиента найдено в списке клиентов - далее нужно прописать месяц, сумму и номер услуги. Наличие клиента в таблице определяется по номеру телефона.

*Ввести расходы*

Экран ввода расхода должен содержать приглашение пользователю на ввод имени получателя (или названия организации), суммы оплаты, дня и месяца, в который производится оплата, бюджетной категории. Затем программа создает новую строку, содержащую эту информацию, и вставляет ее в таблицу расходов.

*Вывести список клиентов*

Программа выводит на экран список клиентов, каждая строка списка состоит из двух полей: номер телефона и имени клиента.

*Вывести список проданных товаров/услуг*

Каждая строка таблицы, которую выводит программа, состоит из имени клиента, оплаченной суммы, типа услуги/товара и номера месяца продажи.

*Вывести таблицу расходов*

Выводится таблица, содержащая строки с информацией о расходах. В одной строке, выводимой программой, находится месяц, день, получатель, сумма и бюджетная категория платежа.

*Вывести годовой отчет*

Программа выводит годовой отчет, состоящий из:

* суммы всех платежей за год;
* списка всех расходов с указанием бюджетной категории;
* общей суммой расходов;
* результирующего годового баланса (доходы/убытки).

1. **Варианты использования, описанные диаграммами действий**

Диаграммы действий представлены в Приложении, рис. 2-4.

1. **Предварительный список существительных (потенциальные классы программы)**

* Экран интерфейса пользователя.
* Клиент.
* Экран ввода клиентов.
* Имя клиента.
* Номер телефона клиента.
* Строка клиента.
* Список клиентов.
* Оплата товара/услуги.
* Экран ввода проданного товара/услуги.
* Месяц.
* Сумма платежа.
* Таблица доходов от продаж.
* Строка платежа.
* Расход.
* Экран ввода расходов.
* Получатель.
* Размер платежа.
* День.
* Бюджетная категория.
* Строка в таблице расходов.
* Таблица расходов.
* Годовой отчет.
* Сумма клиентских платежей.
* Суммарные расходы по категориям.
* Чистая прибыль.

1. **Уточненный список существительных. Пояснение предназначения каждого класса**

* Экран интерфейса пользователя.

UserInterface. Отвечает за взаимодействия пользователя с программой.

* Клиент.

Client. Хранит данные об одном клиенте.

* Экран ввода клиентов.

ClientInputScreen. Класс отвечает за вывод «экрана», куда пользователь вносит данные о новом клиенте.

* Список клиентов.

ClientList. Список всех введенных клиентов. Взаимодействует с классом Client.

* Экран ввода проданного товара/услуги.

SumInputScreen. Отображает «экран» для введения информации о приобретенной клиентом услуге/товаре.

* Строка платежа.

PayRow. Содержит информацию об одном платеже.

* Таблица доходов от продаж.

PayRecord. Содержит информацию обо всех платежах, взаимодействует с PayRow.

* Затраты.

Expense. Хранит данные о единице расходов.

* Таблица расходов.

ExpensePecord. Содержит информацию обо всех расходах. Взаимодействует с PayRow.

* Экран ввода расходов.

ExpenseInputScreen. Отображает «экран» ввода расходов.

* Годовой отчет.

AnnualReport. Отображает все расходы и доходы, а также чистую прибыль. Взаимодействует с PayRecord и ExpenseRecord.

1. **Список атрибутов по каждому классу**

Экран интерфейса пользователя.

* Указатель на список клиентов;
* Указатель на таблицу доходов;
* Указатель на таблицу расходов;
* Указатель на экран ввода клиентов;
* Указатель на экран ввода проданного товара/услуги;
* Указатель на экран ввода расходов;
* Указатель на годовой отчет.

Клиент.

* Имя;
* Номер телефона.

Экран ввода клиентов.

* Указатель на список клиентов;
* Имя;
* Номер телефона.

Список клиентов.

* Указатель на класс клиентов;
* Итератор.

Экран ввода проданного товара/услуги.

* Указатель на список клиентов;
* Указатель на таблицу доходов;
* Имя;
* Сумма;
* Месяц;
* Тип услуги.

Таблица доходов от продаж.

* Указатель на строку платежа;
* Итератор.

Строка платежа.

* Имя;
* Сумма;
* Номер месяца;
* Тип услуги.

Затраты.

* День;
* Месяц;
* Сумма;
* Категория расходов;
* Получатель платежа.

Экран ввода расходов.

* Указатель на таблицу расходов.

Таблица расходов.

* Указатели на расходы;
* Итератор.

Годовой отчет.

* Сума доходов;
* Сумма расходов;
* Указатель на таблицу доходов;
* Указатель на таблицу расходов.

1. **Список сообщений, которыми будут обмениваться классы**

*«Начать программу»*

Когда запускается программа, на экран должно выводиться меню, из которого пользователь может выбрать нужное действие.

Происходит обращение к экрану интерфейса, который выводит на экран сообщение с просьбой выбрать нужную операцию из предложенных.

*«Добавить клиента»*

Экран интерфейса обращается к экрану ввода клиентов и вызывает его метод setClient. Данный метод запрашивает у пользователя ввод имени и номера телефона клиента, а затем приказывает создать новый объект класса Client с указанными атрибутами и занести его в список клиентов (в его конец) через метод insertClient.

*«Ввести проданную услугу/товар»*

Экран ввода услуги/товара запрашивает имя клиента. По имени клиента через метод getphNum из списка клиентов возвращается номер телефона клиента. Полученная строка проверяется на пустоту. Если возвращенная строка пуста, значит, такого клиента в списке не существует, выводится сообщение об отсутствии клиента в списке.

Если возвращенная строка не пуста, значит, клиент существует. Далее экраном ввода услуги/товара выводится сообщение с просьбой ввести сумму платежа, номер месяца оплаты и тип услуги. Далее вызывается метод insertPay. Данный метод создает новый объект класса PayRow (строка платежа) и через метод setPay записывает введенные значения в созданный объект. Строка платежа создана.

*«Добавить расходы»*

Экран ввода расходов запрашивает у пользователя на ввод день, месяц, категорию расходов, получателя и сумму. Далее создается новый объект класса Expense (расходы) с указанными атрибутами. Через метод insertExp указатель на него заносится в таблицу расходов.

*«Вывести список клиентов»*

Экран интерфейса вызывает метод display класса ClientList. Данный метод сначала выводит на экран «шапку» таблицы, а затем проверяет, содержит ли список клиентов какие-то данные. Если список пуст, на экран выводится сообщение «Нет клиентов».

Если в списке есть клиенты, то с помощью итератора и методов getPhoneNumber и getName на экран выводятся номер телефона и имя каждого хранящегося в списке клиента.

*«Вывести проданные товары и услуги»*

Экран интерфейса вызывает метод display класса PayRecord. Данный метод сначала выводит на экран «шапку» таблицы, а затем проверяет, содержит ли список оплаты услуг какие-то записи. Если список пуст, на экран выводится сообщение «Нет платежей».

Если в Списке есть клиенты, то с помощью итератора и методов getService, getMonth, getPayNo и getName на экран выводятся имя каждого хранящегося в списке клиента, тип услуги/товара, сумма покупки и номер месяца покупки.

*«Вывести расходы»*

Экран интерфейса вызывает метод display класса ExpenseRecord. Данный метод сначала выводит на экран «шапку» таблицы, а затем проверяет, содержит ли список расходов какие-то записи. Если список пуст, на экран выводится сообщение «Расходов нет».

Если в Списке есть расходы, то с помощью итератора на экран выводятся месяц оплаты расходов, день, получатель оплаты, сумма и категория расходов.

*«Вывести годовой отчет»*

Экран интерфейса создает новый объект класса AnnualReport и вызывает метод display. Данный метод выводит «шапку» таблицы. Затем через метод getSumOfPays из списка оплаты услуг достаются и суммируются все платежи (и сохраняются в переменную) – и выводятся на экран.

Также, через метод displaySummary класса ExpenseRecord выводятся категория и сумма расходов, суммируются все существующие расходы, сохраняются в переменную – и выводятся на экран.

Затем метод AnnualReport вычитает сумму расходов из суммы доходов и выводит ее на экран.

1. **Диаграмма классов**

Диаграмма классов представлена в Приложении, рис. 5.

1. **Диаграммы последовательностей для нескольких вариантов использования**

Диаграммы последовательностей представлены в Приложении, рис. 6-8.

1. **Написание заголовочных файлов**

//AnnualReport.h

#pragma once

//класс AnnualReport

//Класс годового отчета

class AnnualReport

{

private:

PayRecord\* ptrRR; // записи доходов

ExpenseRecord\* ptrER; // записи расходов

float expenses, pays; // суммы доходов и расходов

public:

AnnualReport(PayRecord\*, ExpenseRecord\*);

void display(); // отображение годового отчета

};

//конец AnnualReport.h

//Client.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

// класс Client (клиенты)

//Он хранит имя клиента и его номер телефона.

class Client

{

private:

string name; // имя клиента

string phoneNumber; // номер телефона клиента

public:

Client(string n, string cl);

~Client();

string getPhoneNumber(); //возвращает номер телефона клиента

string getName(); //возвращает имя клиента

};

//конец Client.h

//ClientInputScreen.h

#pragma once

#include <iostream>

using namespace std;

//класс ClientInputScreen

//класс ClientInputScreen. Это класс, отвечающий за отображение «экрана»,

//куда пользователь может ввести данные о новом клиенте:

class ClientInputScreen

{

private:

ClientList\* ptrClientList;

string tName;

string clientPhone;

public:

ClientInputScreen(ClientList\* ptrTL) : ptrClientList(ptrTL)

{

/\* тут пусто \*/

}

void setClient(); // добавить данные о клиенте

};

//конец ClientInputScreen.h

//ClientList.h

#pragma once

using namespace std;

//класс ClientList

//класс ClientList — список всех клиентов.

//Он содержит множество указателей на класс Client и оперирует ими при выводе

class ClientList

{

private:

// установить указатели на клиентов

list <Client\*> setPtrsClients; // указатели на класс клиентов

list <Client\*>::iterator iter; //итератор

public:

~ClientList(); // деструктор (удаление клиентов)

void insertClient(Client\*); // добавить клиента в список

string getphNum(string); // возвращает номер телефона

void display(); // вывод списка клиентов

};

//конец ClientList.h

//Expense.h

#pragma once

//класс Expense

//Класс затрат

class Expense

{

public:

int month;

int day; // месяц и день уплаты расходов

string category; // категория расходов (налоги, аренда, услуги гробовщиков и т.д.)

string payee; // кому платим

float amount; // сколько платим

Expense() :

month(0),

day(0),

amount(0)

{ }

Expense(int m, int d, string c, string p, float a) :

month(m), day(d), category(c), payee(p), amount(a)

{

/\* тут пусто! \*/

}

};

//конец Expense.h

//ExpenseInputScreen.h

#pragma once

//класс ExpenseInputScreen

//Класс для ввода расходов

class ExpenseInputScreen

{

private:

ExpenseRecord\* ptrExpenseRecord; // запись о расходах

public:

ExpenseInputScreen(ExpenseRecord\*);

void setExpense();

};

//конец ExpenseInputScreen.h

//ExpenseRecord.h

#pragma once

//класс ExpenseRecord

//Класс записей о затратах

class ExpenseRecord

{

private:

vector<Expense\*> vectPtrsExpenses; //вектор указателей на расходы

vector<Expense\*>::iterator iter;

public:

~ExpenseRecord();

void insertExp(Expense\*);

void display();

float displaySummary(); // нужно для годового отчета

};

//конец ExpenseRecord.h

//Header.h

//заголовочный файл содержит объявления классов и т.п. (к нему подключены все заголовочные файлы)

#include <iostream>

#include <vector>

#include <list>

#include <string>

#include "Client.h"

#include "ClientList.h"

#include "ClientInputScreen.h"

#include "PayRow.h"

#include "PayRecord.h"

#include "SumInputScreen.h"

#include "Expense.h"

#include "ExpenseRecord.h"

#include "ExpenseInputScreen.h"

#include "AnnualReport.h"

#include "UserInterface.h"

using namespace std;

// ГЛОБАЛЬНЫЕ МЕТОДЫ

void getaLine(string& inStr); // получение строки текста

char getaChar(); // получение символа

void checkDigit(int& x); //проверка, что введено число

//конец Header.h

//PayRecord.h

#pragma once

// класс PayRecord

//класс PayRecord. Он хранит непосредственно записи об оплате услуг.

//С ним будет взаимодействовать экран добавления оплаты.

class PayRecord

{

private:

list <PayRow\*> setPtrsRR; // указатели на объекты PayRow (по одному на клиента)

list <PayRow\*>::iterator iter;

public:

~PayRecord();

void insertPay(string, string, int, float, int); // добавить платеж

void display(); // отобразить все строки с платежами

float getSumOfPays(); // подсчитать сумму всех платежей всех клиентов

};

//конец PayRecord.h

//PayRow.h

#pragma once

//класс PayRow

//класс, хранящий одну табличную строку доходов (оплаченной услуги)

// одна строка таблицы прибыли: имя клиента, тип услуги, месяц сделки, сумма

class PayRow

{

private:

string clName; // имя клиента

float pay = 0; // плата

int service = 0; //тип услуги под номером 1, 2, 3 или 4

int month = 0; // номер месяца

public:

PayRow(string); // конструктор с одним параметром

void setPay(string, string, int, float, int); // добавить плату

string getName(); //Запрос имени

float getPayNo(); //Запрос платы

string getService(); //запрос услуги

int getMonth(); //запрос месяца

};

//конец PayRow.h

//SumInputScreen.h

#pragma once

//класс SumInputScreen

//Экран для добавления платежа

class SumInputScreen

{

private:

ClientList\* ptrClientList; // список клиентов

PayRecord\* ptrPayRecord; // список записей об оплате услуг

string clientName; // имя клиента, который оплатил услугу

float sumPaid = 0; // сумма платежа

int month = 0; // за месяц

string phoneNo; // номер телефона

int service = 0; // тип услуги

public:

SumInputScreen(ClientList\* ptrTL, PayRecord\* ptrRR) : ptrClientList(ptrTL),

ptrPayRecord(ptrRR)

{

/\*тут пусто\*/

}

void setPay(); // добавить оплату одной услуги клиентом

};

//конец SumInputScreen.h

//UserInterface.h

#pragma once

// Класс UserInterface

//Главный класс для запуска приложения:

//этот класс определяет взаимодействие юзера с программой.

class UserInterface

{

private:

ClientList\* ptrClientList;

ClientInputScreen\* ptrClientInputScreen;

PayRecord\* ptrPayRecord;

SumInputScreen\* ptrPayInputScreen;

ExpenseRecord\* ptrExpenseRecord;

ExpenseInputScreen\* ptrExpenseInputScreen;

AnnualReport\* ptrAnnualReport;

char ch;

public:

UserInterface();

~UserInterface();

void interact();

};

// конец userInterface.h

1. **Написание исходных файлов**

//AnnualReport.cpp

#include "Header.h"

//методы класса AnnualReport

//Конструктор

AnnualReport::AnnualReport(PayRecord\* pRR, ExpenseRecord\* pER) : ptrRR(pRR), ptrER(pER)

{ /\* пусто \*/

}

//---------------------------------------------------------

void AnnualReport::display()

{

cout << "Годовой отчет\n--------------\n" << endl;

cout << "Доходы\n" << endl;

cout << "\tПлатежи клиентов:\t\t";

pays = ptrRR->getSumOfPays(); //помещаем сумму доходов в переменную

cout << pays << endl;

cout << "Расходы\n" << endl;

expenses = ptrER->displaySummary(); //помещаем сумму всех затрат в переменную

cout << "Расходы всего:\t\t";

cout << expenses << endl;

// вычисляем прибыльность

cout << "\nЧистая прибыль:\t\t\t" << (pays - expenses) << endl;

}

//конец AnnualReport.cpp

//Client.cpp

#include "Header.h"

//методы класса Client

//в конструкторе задаём имя клиента и номер телефона

Client::Client(string n, string cl) : name(n), phoneNumber(cl)

{

/\* тут пусто \*/

}

//---------------------------------------------------------

Client::~Client() // деструктор

{

/\* тут тоже пусто \*/

}

//---------------------------------------------------------

string Client::getPhoneNumber() // возвращает номер телефона клиента

{

return phoneNumber;

}

//--------------------------------------------------------

string Client::getName() // возвращает имя клиента

{

return name;

}

//конец Client.cpp

//ClientInputScreen.cpp

#include "Header.h"

//метод класса ClientInputScreen

void ClientInputScreen::setClient() // добавить данные о клиенте

{

cout << "Введите имя клиента (Иванов И И): " << endl;

getaLine(tName);

cout << "Введите номер телефона клиента (89008753626): " << endl;

getaLine(clientPhone);

Client\* ptrClient = new Client(tName, clientPhone); // создать клиента

ptrClientList->insertClient(ptrClient); // занести в список клиентов

}

//конец ClientInputScreen.cpp

//ClientList.cpp

#include "Header.h"

//методы класса ClientList

ClientList::~ClientList() // деструктор

{

while (!setPtrsClients.empty()) // удаление всех клиентов,

{ // удаление указателей из контейнера

iter = setPtrsClients.begin();

delete\* iter;

setPtrsClients.erase(iter);

}

}

//---------------------------------------------------------

void ClientList::insertClient(Client\* ptrT)

{

setPtrsClients.push\_back(ptrT); // вставка нового клиента в список

}

//---------------------------------------------------------

string ClientList::getphNum(string tName) // получить номер телефона по имени клиента

{

string phoneNo;

iter = setPtrsClients.begin();

while (iter != setPtrsClients.end())

{ // поиск клиента в списке (достаем у каждого клиента номер телефона)

phoneNo = (\*iter)->getPhoneNumber();

if (tName == ((\*iter)->getName())) // сравниваем по именам и

{

// если получившаяся пара совпадает - значит,

//если запись об этом клиенте найдена в списке клиентов, в этом случае

return phoneNo; // возвращаем номер телефона

}

iter++;

}

return string(); // если нет - возвращаем пустую строку

}

//--------------------------------------------------------

void ClientList::display() // вывод списка клиентов

{

cout << "\nНомер телефона\tИмя клиента\n-------------------\n";

if (setPtrsClients.empty()) // если список клиентов пуст

cout << "\*\*\*Нет клиентов\*\*\*\n" << endl; // выводим запись, что он пуст)

else

{

iter = setPtrsClients.begin();

while (iter != setPtrsClients.end()) // распечатываем всех клиентов

{

cout << (\*iter)->getPhoneNumber() << " || " << (\*iter)->getName() << endl;

\*iter++;

}

}

}

//конец ClientList.cpp

//ExpenseInputScreen.cpp

#include "Header.h"

//методы класса ExpenseInputScreen

// конструктор

ExpenseInputScreen::ExpenseInputScreen(ExpenseRecord\* per) : ptrExpenseRecord(per)

{

/\*пусто\*/

}

//------------------------------------------------------

void ExpenseInputScreen::setExpense()

{

int month, day;

string category, payee;

float amount;

cout << "Введите месяц (1-12): ";

checkDigit(month);

cin.ignore(80, '\n');

cout << "Введите день (1-31): ";

checkDigit(day);

cin.ignore(80, '\n');

cout << "Введите категорию расходов (Налоги, ремонт, аренда, крематорий и т.д.): ";

getaLine(category);

cout << "Введите получателя (ИП Могилки): ";

getaLine(payee);

cout << "Введите сумму (3900.99): ";

cin >> amount;

cin.ignore(80, '\n');

// создаем новый расход

Expense\* ptrExpense = new Expense(month, day, category, payee, amount);

// вставляем расход в список всех расходов

ptrExpenseRecord->insertExp(ptrExpense);

}

//конец ExpenseInputScreen.cpp

//ExpenseRecord.cpp

#include "Header.h"

//методы класса expenseRecord

ExpenseRecord::~ExpenseRecord() // деструктор

{ // удалить объекты expense

// удалить указатели на вектор

while (!vectPtrsExpenses.empty())

{

iter = vectPtrsExpenses.begin();

delete\* iter;

vectPtrsExpenses.erase(iter);

}

}

//--------------------------------------------------------

void ExpenseRecord::insertExp(Expense\* ptrExp)

{

vectPtrsExpenses.push\_back(ptrExp);

}

//---------------------------------------------------------

void ExpenseRecord::display() // распечатываем все расходы

{

cout << "\nДата\tПолучатель\tСумма\tКатегория\n"

<< "----------------------------------------\n" << endl;

if (vectPtrsExpenses.size() == 0) // В контейнере нет расходов

cout << "\*\*\*Расходов нет\*\*\*\n" << endl;

else

{

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{ // распечатываем все расходы

cout << (\*iter)->month << '/' << (\*iter)->day << '\t' << (\*iter)->payee << '\t' << '\t';

cout << (\*iter)->amount << '\t' << (\*iter)->category << endl;

iter++;

}

cout << endl;

}

}

//--------------------------------------------------------

// используется при составлении годового отчета

float ExpenseRecord::displaySummary()

{

float totalExpenses = 0; // Сумма по всем категориям расходов

if (vectPtrsExpenses.size() == 0)

{

cout << "\tВсе категории\t0\n";

return 0;

}

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{

//выводим на экран категории расходов

cout << '\t' << ((\*iter)->category) << '\t' << ((\*iter)->amount) << endl;

totalExpenses += (\*iter)->amount; //подсчитываем все расходы

iter++;

}

return totalExpenses;

}

//конец ExpenseRecord.cpp

// Main.cpp

#include "Header.h"

#include <iostream>

#include <vector>

#include <list>

#include <string>

int main()

{

//Функция setlocale() задаёт локализацию программы.

setlocale(LC\_ALL, "rus");

UserInterface theUserInterface;

theUserInterface.interact();

return 0;

}

//конец Main.cpp

//PayRecord.cpp

#include "Header.h"

//методы класса PayRecord

PayRecord::~PayRecord() // деструктор

{ // удалить строки с платежами,

// удалить указатели из множества.

while (!setPtrsRR.empty())

{

iter = setPtrsRR.begin();

delete\* iter;

setPtrsRR.erase(iter);

}

}

//---------------------------------------------------------

void PayRecord::insertPay(string name, string phNo, int month, float amount, int serv)

{

PayRow\* ptrRow = new PayRow(name);

ptrRow->setPay(name, phNo, month, amount, serv); // заносим в нее платеж

setPtrsRR.push\_back(ptrRow); // заносим строку в вектор

}

//---------------------------------------------------------

void PayRecord::display() // отобразить все строки с доходами

{

cout << "\nИмя клиента\tТип услуги\tМесяц\t Сумма\n" << endl

<< "------------------------------------------------------------------\n" << endl;

if (setPtrsRR.empty())

cout << "\*\*\*Нет платежей!\*\*\*\n" << endl;

else

{

iter = setPtrsRR.begin(); //итератор на список с указателями на объекты PayRow

while (iter != setPtrsRR.end())

{

cout << (\*iter)->getName() << '\t'; // вывести имя клиента

cout << (\*iter)->getService() << "\t";

cout << (\*iter)->getMonth() << "\t";

cout << (\*iter)->getPayNo() << " ";

cout << endl;

iter++;

}

cout << endl;

cout << endl;

}

}

//---------------------------------------------------------

float PayRecord::getSumOfPays() // сумма всех платежей

{

float sumRents = 0.0;

iter = setPtrsRR.begin();

while (iter != setPtrsRR.end())

{ // плюсуем суммы всех платежей клиентов за все время

sumRents += (\*iter)->getPayNo();

iter++;

}

return sumRents;

}

//конец PayRecord.cpp

//PayRow.cpp

#include "Header.h"

//методы класса PayRow//

PayRow::PayRow(string an) : clName() //конструктор

{ //конструктор с одним параметром

}

//---------------------------------------------------------

void PayRow::setPay(string cl, string ph, int m, float am, int serv) // сеттер оплата за месяц m, сумма - am

{

//инициализаци переменных

pay = am;

clName = cl;

month = m;

service = serv;

}

//---------------------------------------------------------

string PayRow::getName() // геттер запрос имени клиента

{

return clName;

}

//---------------------------------------------------------

float PayRow::getPayNo() //Геттер запрос платежа

{

return pay;

}

//------------------------------------------------------------

int PayRow::getMonth() //геттер запрос месяца

{

return (month + 1); //+1 из-за специфики счета: считается с 0

}

//-------------------------------------------------------------

string PayRow::getService() //Геттер запрос услуги

{

switch (service)

{

case 1:

return "Подготовка покойника";

break;

case 2:

return "Продажа гроба ";

break;

case 3:

return "Копка могилы ";

break;

case 4:

return "Услуги гробовщика ";

break;

default:

return "Другое ";

break;

}

}

//конец PayRow.cpp

//Source.cpp

#include "Header.h"

void getaLine(string& inStr) // получение строки текста

{

char temp[21];

cin.get(temp, 20, '\n');

cin.ignore(20, '\n'); //число пропускаемых символов и символ разделения

inStr = temp;

}

//---------------------------------------------------------

char getaChar() // получение символа

{

char ch = cin.get();

cin.ignore(80, '\n'); //число пропускаемых символов и символ разделения

return ch;

}

void checkDigit(int& x) //проверка на ввод целого числа, чтобы программа не вылетала

{

cin >> x;

while (cin.fail()) {

cout << "Введите число!" << std::endl;

cin.clear();

cin.ignore(256, '\n');

cin >> x;

}

}

//конец Source.cpp

//SumInputScreen.cpp

#include "Header.h"

void SumInputScreen::setPay() //ввод и запись платежа в таблицу

{

cout << "Введите имя клиента: ";

getaLine(clientName);

// получить номер телефона по имени клиента

phoneNo = ptrClientList->getphNum(clientName);

if (phoneNo.empty() == false) // если клиент найден -

{ // получить сумму платежа

cout << "Введите сумму платежа (2000.37): " << endl;

cin >> sumPaid; // вводим сумму платежа

cin.ignore(80, '\n');

cout << "Введите номер месяца оплаты (1-12): " << endl;

checkDigit(month);

cin.ignore(80, '\n');

month--; // (внутренняя нумерация 0-11)

cout << "Введите тип услуги (Подготовка покойника (1), Продажа гроба(2), Копка могилы (3), Услуги гробовщика (4)): " << endl;

checkDigit(service); //ввод услуги с проверкой int

cin.ignore(80, '\n');

// вставляем данные в запись об оплате

ptrPayRecord->insertPay(clientName, phoneNo, month, sumPaid, service);

}

else

cout << "Такого клиента нет.\n" << endl;

} //конец SumInputScreen.cpp

//UserInterface.cpp

#include "Header.h"

//методы класса userInterface

UserInterface::UserInterface()

{

ptrClientList = new ClientList;

ptrPayRecord = new PayRecord;

ptrExpenseRecord = new ExpenseRecord;

}

//---------------------------------------------------------

UserInterface::~UserInterface()

{

delete ptrClientList;

delete ptrPayRecord;

delete ptrExpenseRecord;

}

//---------------------------------------------------------

void UserInterface::interact()

{

while (true)

{

cout << " ВВЕСТИ ДАННЫЕ 'i', \n"

<< " ВЫВЕСТИ ОТЧЕТ 'd', \n"

<< " ВЫХОД 'q': \n";

ch = getaChar();

if (ch == 'i') // ввод данных

{

cout << " Добавить клиента 't', \n"

<< " Записать проданную услугу/товар 'r', \n"

<< " Добавить расходы 'e': \n";

ch = getaChar();

switch (ch)

{

//экраны ввода существуют только во время их

//использования

case 't': ptrClientInputScreen =

new ClientInputScreen(ptrClientList);

ptrClientInputScreen->setClient();

delete ptrClientInputScreen;

break;

case 'r': ptrPayInputScreen =

new SumInputScreen(ptrClientList, ptrPayRecord);

ptrPayInputScreen->setPay();

delete ptrPayInputScreen;

break;

case 'e': ptrExpenseInputScreen =

new ExpenseInputScreen(ptrExpenseRecord);

ptrExpenseInputScreen->setExpense();

delete ptrExpenseInputScreen;

break;

default: cout << "Неизвестная функция\n";

break;

} // конец секции switch

} // конец условия if

else if (ch == 'd') // вывод данных

{

cout << " вывести список клиентов 't', \n"

<< " Вывести проданные тавары и услуги 'r' \n"

<< " Вывести расходы 'e', \n"

<< " Вывести годовой отчет 'a': \n";

ch = getaChar();

switch (ch)

{

case 't': ptrClientList->display();

break;

case 'r': ptrPayRecord->display();

break;

case 'e': ptrExpenseRecord->display();

break;

case 'a':

ptrAnnualReport = new AnnualReport(ptrPayRecord,

ptrExpenseRecord);

ptrAnnualReport->display();

delete ptrAnnualReport;

break;

default: cout << "Такой функции нет\n";

break;

} // конец switch

} // конец elseif

else if (ch == 'q')

return; // выход

else

cout << "Такой функции нет. Нажимайте только 'i', 'd' или 'q'\n";

} // конец while

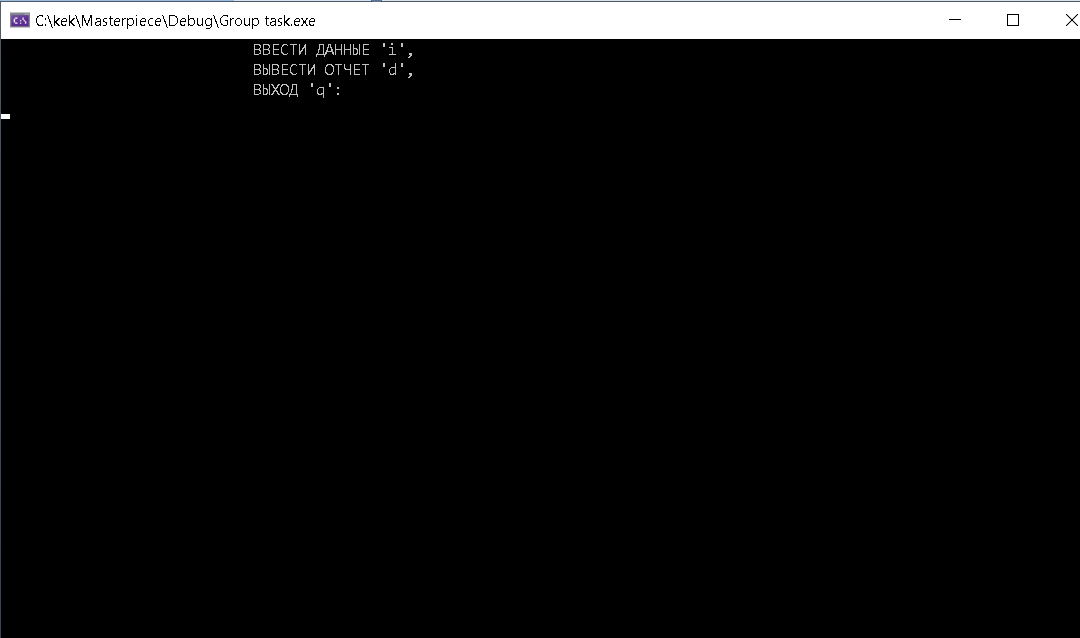
} // конец interact()

//конец UserInterface.cpp

1. **Руководство пользователя**

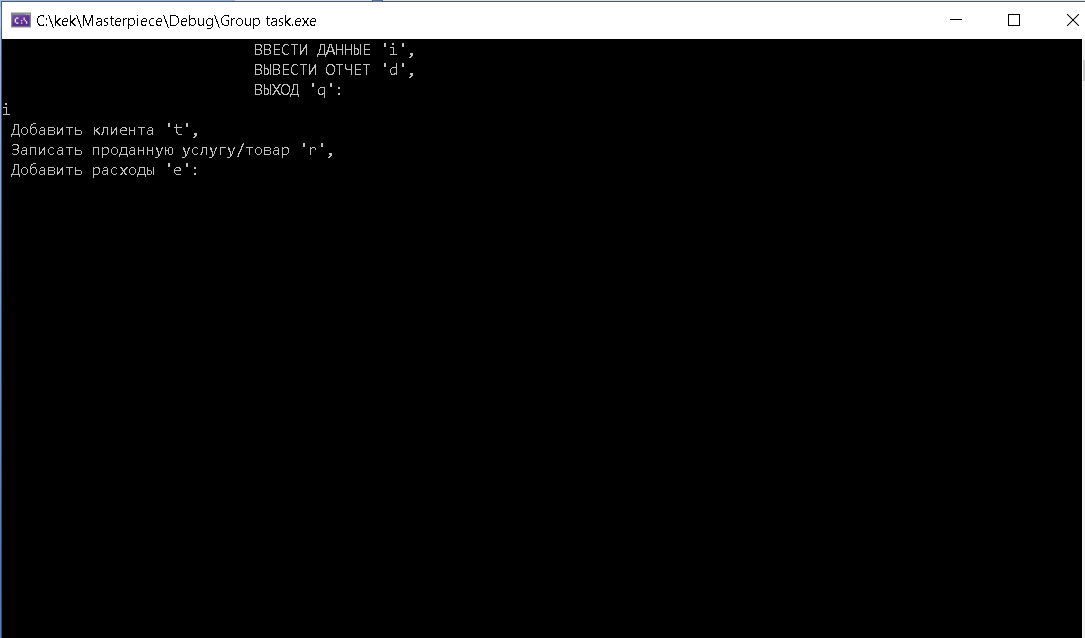
При запуске программы пользователь первым делом видит экран интерфейса, который отображает три операции.

1. Операция «ВВЕСТИ ДАННЫЕ, ‘i’». Для ее выбора нужно нажать ключ i, и программа отобразит следующие доступные операции ввода данных.
2. Операция «ВЫВЕСТИ ОТЧЕТ, ‘d’». Для выбора нужно нажать ключ d, и программа отразит следующие доступные операции вывода данных, уже записанных в программу.
3. Операция «ВЫХОД, ‘q’». Для выбора нужно нажать ключ q, и программа завершит программу, после чего нужно нажать любую клавишу для закрытия окна консоли.

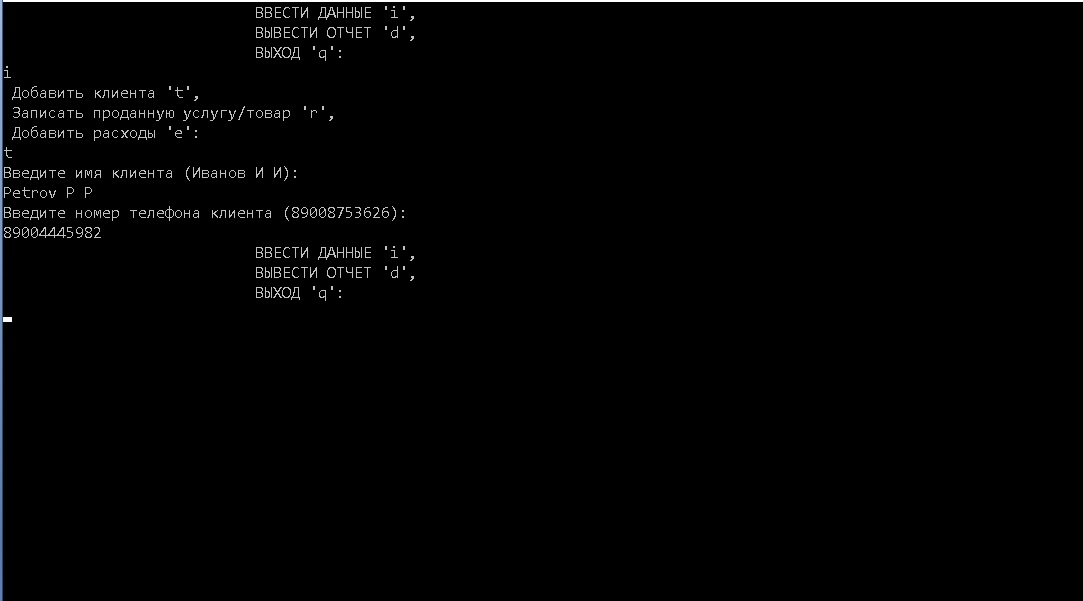


Ввод данных.

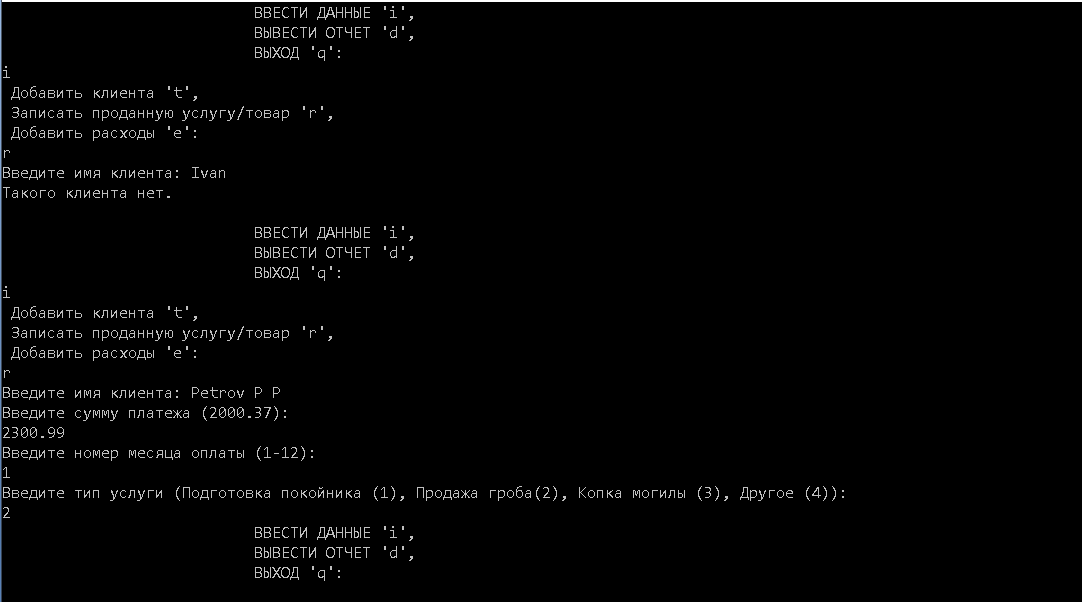
После выбора операции ввода данных, у пользователя на экране появятся следующие 3 операции: Добавить клиента, записать проданную услугу/товар, записать расходы.



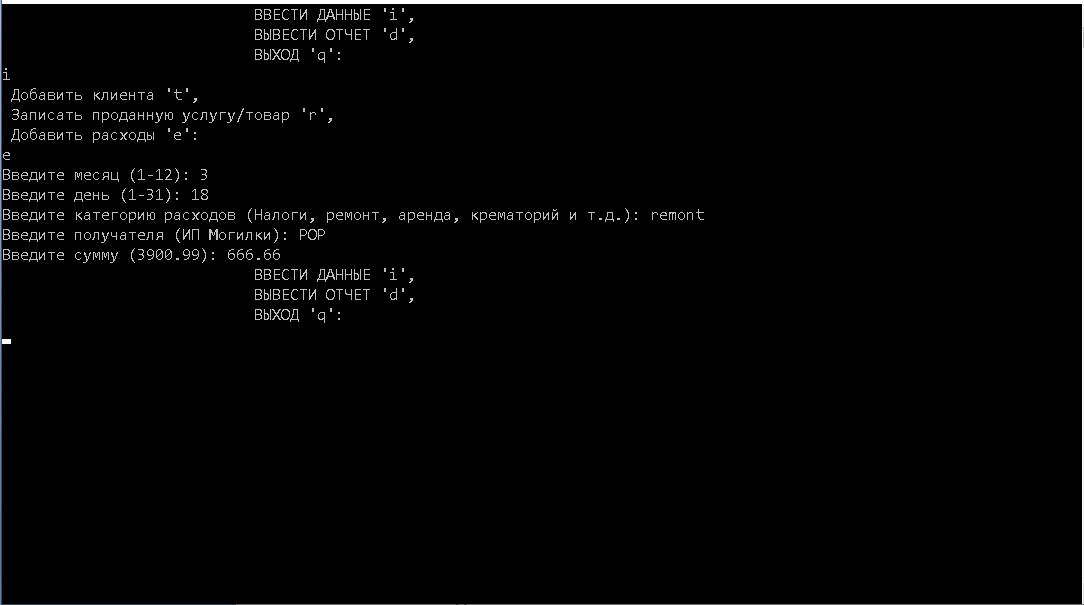
Чтобы выбрать добавление клиента, нужно нажать ключ t. В таком случае программа запросит на ввод имя клиента (пример ввода имени прописан в скобках), а затем его номер телефона, которые пользователь должен ввести с помощью клавиатуры. Для подтверждения ввода нужно нажать Enter. После завершения ввода данных о клиенте программа снова выведет первые три операции.



Добавить товар или услугу можно только в том случае, если покупатель уже существует в списке клиентов. Чтобы выбрать добавление товара/услуги, нужно нажать ключ r. После нажатия ключа программа попросит ввести имя клиента (он должен существовать, иначе будет выведено сообщение об отсутствии клиента), сумму покупки, месяц совершения сделки и тип услуги/товара (нужно выбрать цифру, которая записана в скобках после каждого товара). Пользователь должен ввести все вышеперечисленные данные с помощью клавиатуры. Для подтверждения ввода нужно нажать Enter. После завершения ввода данных о покупке программа снова выведет первые три операции.



Чтобы выбрать добавление расходов, нужно нажать ключ e. После нажатия ключа программа попросит ввести месяц, день, категорию расходов, получателя платежа и сумму. Пользователь должен ввести все вышеперечисленные данные с помощью клавиатуры. Для подтверждения ввода нужно нажать Enter. После завершения ввода данных о расходах программа снова выведет первые три операции.

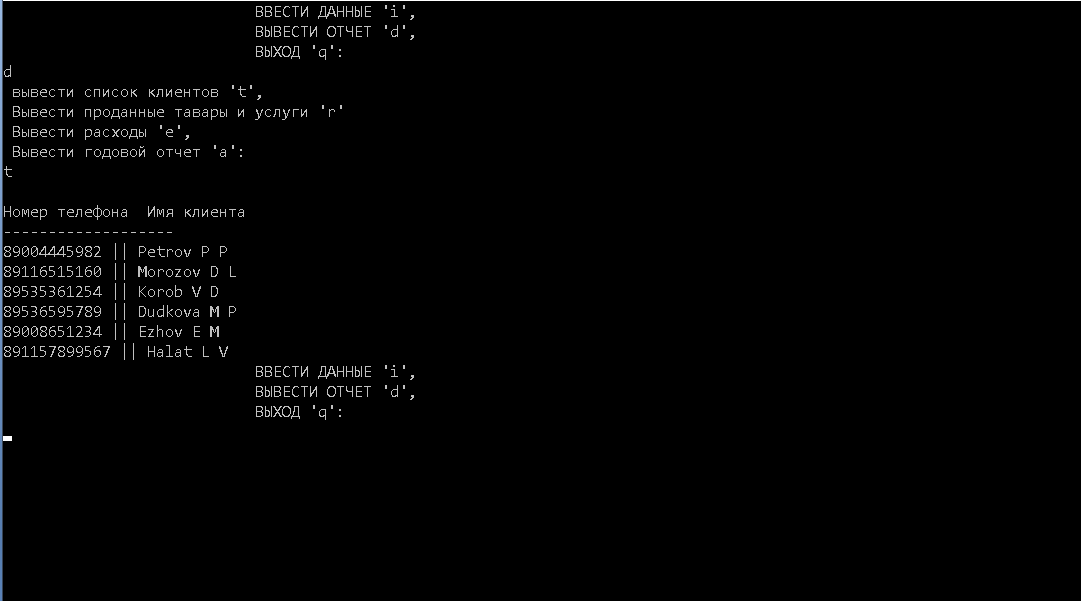


Вывод данных.

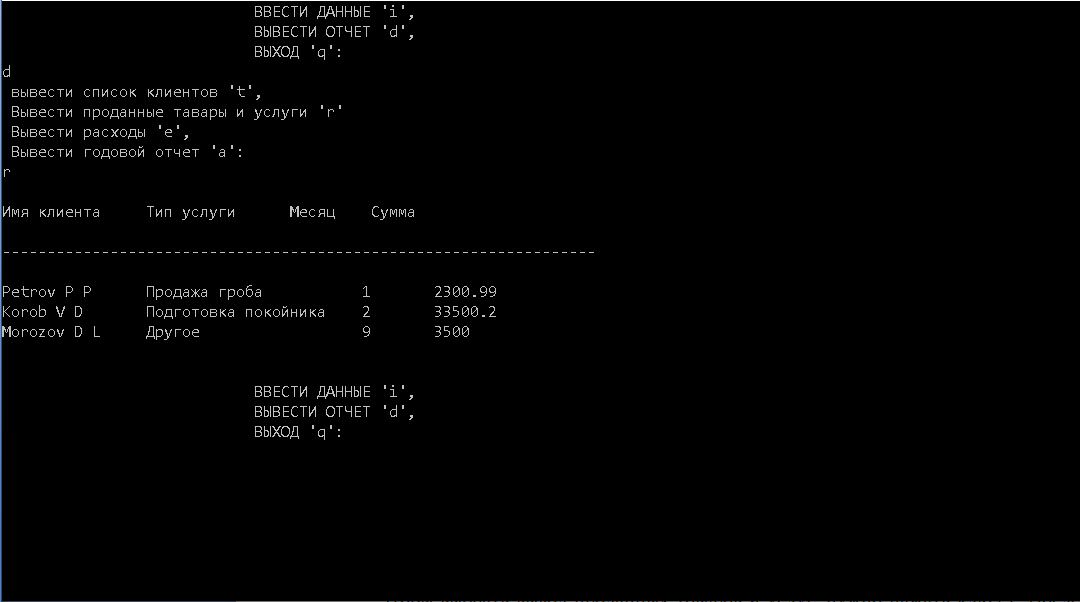
Для вывода каких-либо данных нужно нажать ключ d (ВЫВЕСТИ ОТЧЕТ). Соответственно, если выводить нечего – программа даст пользователю знать об этом.



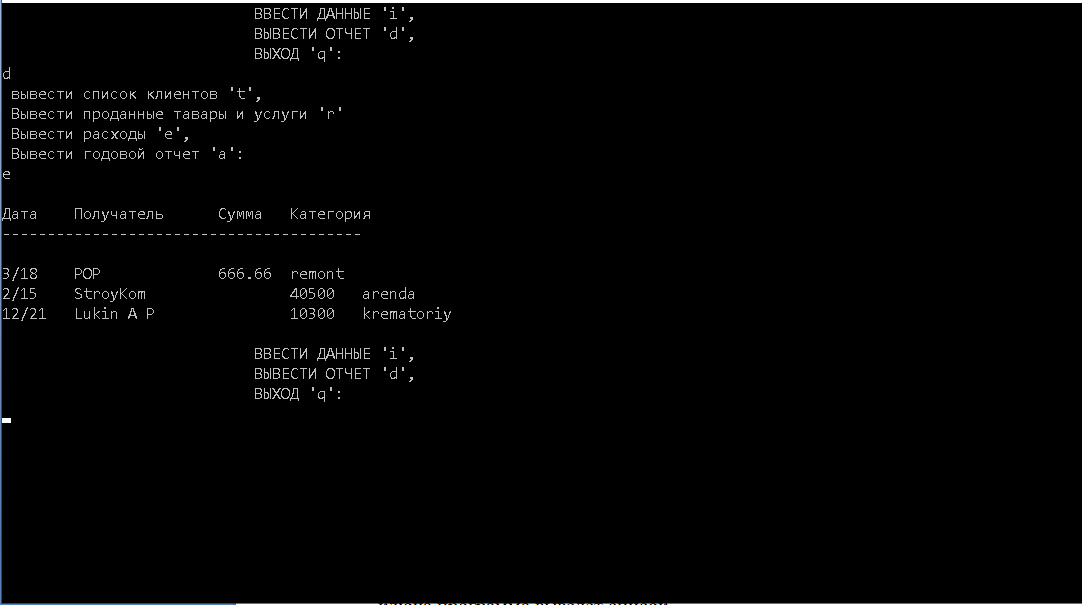
Чтобы выбрать вывод клиентов, нужно нажать ключ t. После нажатия ключа программа выведет список всех существующих клиентов с их номерами телефонов.



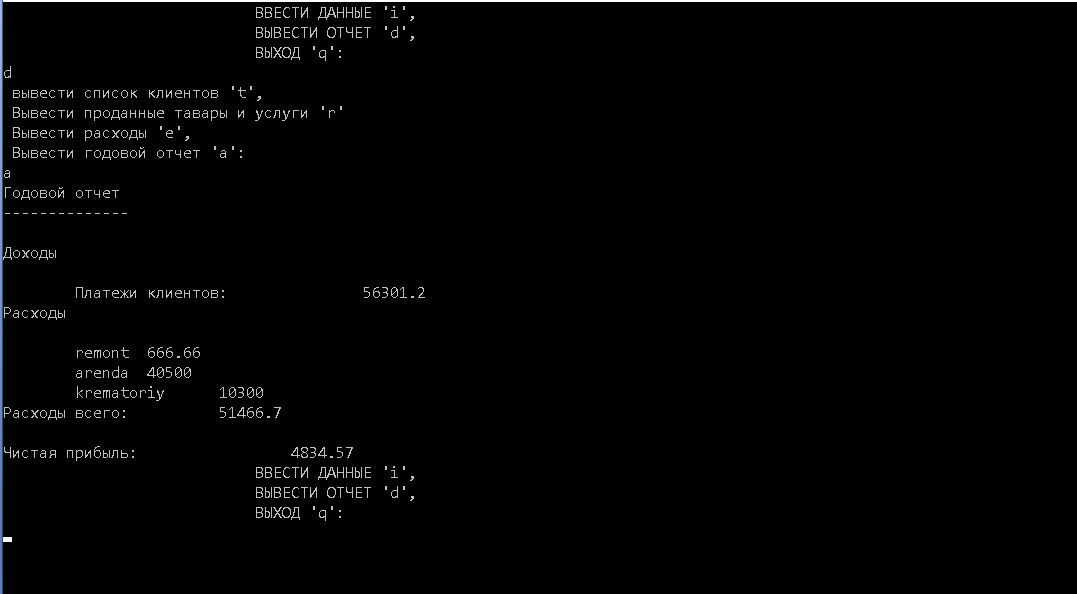
Чтобы выбрать вывод проданных товаров и услуг, нужно нажать ключ r. После нажатия ключа программа выведет список всех осуществленных сделок. На экран выводится имя клиента, тип услуги/товара, месяц сделки и сумма сделки.



Чтобы выбрать вывод оплаченных расходов, нужно нажать ключ e. После нажатия ключа программа выведет список всех расходов с указанием дня и месяца, получателя, суммы и категории.



Чтобы выбрать вывод годового отчета, нужно нажать ключ a. После нажатия ключа программа выведет общую сумму доходов, список расходов и общую сумму расходов. Также, будет выведена чистая прибыль бюро (разность доходов и расходов).



**История проекта на GitHub**

Ссылка на репозиторий: <https://github.com/maroonOcat/Funeral>

*Коммиты на github через git log начиная с последнего:*

1. commit ace1b63e0b3435ef23bfeb62caa73218be07dfc3 (HEAD -> master, origin/master, origin/develop, origin/HEAD)

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 14:07:20 2020 +0300

VS 2013 files added

**Добавлены файлы из папки с проектом Visual Studio 2013.**

1. commit 35ce7063470920f10f4b71f642c90c541f7b2210

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 14:04:00 2020 +0300

Debug folder + VS files

**Добавлена папка Debug.**

1. commit b802b1fd704ea5cfe87e1df6c016dee5a82a70c5

Merge: 99aa86b e9c5aaf

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:59:08 2020 +0300

Merge branch 'mainFunc' into develop

**Ветка, содержащая основную процедуру, влита в develop.**

1. commit 99aa86b24411f47da7eaece6552ab06ce768fec5

Merge: e2e92f1 7139431

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:57:17 2020 +0300

Merge branch 'featureUser' into develop

**Ветка, содержащая хедер и cpp-файл с определением методов класса UserInterface влита в develop.**

1. commit e2e92f14512bea168a5c5223c477320efc564090

Merge: a0c4fe2 c724e15

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:56:39 2020 +0300

Merge branch 'featureExpense' into develop

**Ветка, содержащая хедеры и cpp-файлы с определением методов классов Expense, ExpenseRecord, ExpenseInputScreen влита в develop.**

1. commit e9c5aaf74315f59f9b2b69609a50dea7a401e046 (origin/mainFunc)

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:54:13 2020 +0300

file for compilation

**В ветку develop добавлен файл targetver**

1. commit 5bdc7a1835a4266cec0cf36a5e6df3e4327e982a

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:48:15 2020 +0300

main.cpp modified, source.cpp added

**Исправление файла main.cpp и добавление файла source.cpp с определением глобальных методов.**

1. commit c724e15c125fb92e0790f2938fe562906f3a4b22 (origin/featureExpense)

Merge: c89d8aa 7253583

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:43:44 2020 +0300

Merge branch 'featureExpense' of https://github.com/maroonOcat/Funeral into featureExpense

**Обновление локального репозитория.**

1. commit c89d8aac9162b7a0243b855c0096d5140ba8e33d

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:42:24 2020 +0300

ExpenseRecord.cpp added

**Добавление ExpenseRecord.cpp в ветку featureExpense.**

1. commit a0c4fe29a71d6c92b02e566e23ca226b5a0a56ac

Merge: 88d35e0 7253583

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:40:35 2020 +0300

Merge branch 'featureExpense' of https://github.com/maroonOcat/Funeral into featureClient

**Слияние веток featureExpense и featureClient**

1. commit 7253583b0ca6934cce3084cfc0d372b4a4be4abf

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:39:38 2020 +0300

ExpenseInputScreen.cpp added by Lera

**Лера добавила файл ExpenseInputScreen.cpp в ветку featureExpense.**

1. commit 88d35e0f5fb28dcd2ea4421aa876c45411c8b66c

Merge: 6556243 3f1f4ce

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:36:53 2020 +0300

Merge branch 'featurePay' of https://github.com/maroonOcat/Funeral into featureClient

**Слияние веток featurePay и featureClient.**

1. commit 3f1f4ceec30e2419e499f62b522d686b0ba87b36 (origin/featurePay)

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:36:11 2020 +0300

SumInputScreen.cpp added by Lera

**Добавление файла SumInputScreen.cpp в ветку featurePay.**

1. commit 65562434347a473447865cc76c0120adf5582d85

Merge: 66203cd ed76694

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:36:00 2020 +0300

Merge branch 'featureAnn' of https://github.com/maroonOcat/Funeral into featureClient

**Слияние веток featureAnn и featureClient**

1. commit 172f263fff05711a39c963dff0b6baef7e302a5a

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:34:22 2020 +0300

PayRow.cpp added by Lera

**Добавление файла PayRow.cpp в ветку featurePay.**

1. commit 66203cd342d05a57a08a83abc2822f20e16017b2 (origin/featureClient)

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:33:55 2020 +0300

ClientInputScreen added

**Добавление файла ClientInputScreen.cpp в ветку featureClient.**

1. commit f214c818114d0b0d8b5f3329fa8608b30291dea5

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:32:38 2020 +0300

ClientList.cpp added

**Добавление файла ClientList.cpp в ветку featureClient.**

1. commit 5df3f64b1f98caaa2fa6e1cfd4223c22eeb54419

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:32:25 2020 +0300

PayRecord.cpp added by Lera

**Добавление файла PayRecord.cpp в ветку featurePay.**

1. commit 1c5acc7658ed33db3933eb691e8f81ca8fcf2cbb

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:31:52 2020 +0300

Client.cpp added

**Добавление файла Client.cpp в ветку featureClient.**

1. commit 7139431b574346bafbf55b81a3e512a358ecc57d (origin/featureUser)

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:26:38 2020 +0300

UserInterface.cpp added

**Добавление файла UserInterface.cpp в ветку featureUser.**

1. commit ed76694299e13b8dda9fb6447d222b5866a3e0ce (origin/featureAnn)

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:24:18 2020 +0300

AnnualReport.cpp added by Lera

**Добавление файла AnnualReport.cpp в ветку featureAnn.**

1. commit 73c3eba655ced50f4cac801da3d483310192a488

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:22:48 2020 +0300

header UserInterface added

**Добавление файла UserInterface.h в ветку featureUser.**

1. commit 1d3669ce63e9675182fb61292b9739ebe510eab5

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:17:51 2020 +0300

header AnnualReport added by Lera

**Добавление файла AnnualReport.h в ветку featureAnn.**

1. commit fc052692e2e4efe63d4a00b8ed2174d2657a1060

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 13:10:30 2020 +0300

header Header.h added

**Добавление файла Header.h в ветку mainFunc.**

1. commit 9f2940ed4eef5bfd16ae60a47dcebb58ace4697a

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:05:01 2020 +0300

header ExpenseInputScreen added by Lera

**Добавление файла ExpenseInputScreen.h в ветку featureExpense.**

1. commit 13f21311ed739b5e826e0af06e20a4b0434dfecb

Merge: 6f773f7 0a24b07

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 13:02:15 2020 +0300

Merge branch 'featureExpense' of https://github.com/maroonOcat/Funeral into featureExpense

**Обновление удаленного репозитория (ветки featureExpense).**

1. commit 0a24b073dc580b66a698ea9cd8746bd7563c1575

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 12:58:18 2020 +0300

header ExpenseRecord added

**Добавление файла ExpenseRecord.h в ветку featureExpense.**

1. commit 12b92e205f67fd16217a566fd3f578accb8efe1e

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Fri Dec 25 12:57:37 2020 +0300

header Expense added

**Добавление файла Expense.h в ветку featureExpense.**

1. commit 6f773f77fa98cb1b7f5b957dd3cfdd226798b1a5

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 12:30:13 2020 +0300

header SumInputScreen added by Lera

**Добавление файла SumInputScreen.h в ветку featurePay.**

1. commit f45d7ddb273ba7e1c41417ad797277b789888a6e

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 12:28:14 2020 +0300

header PayRecord added by Lera

**Добавление файла PayRecord.h в ветку featurePay.**

1. commit 939f26b8cdea56a0cbce47314283888a6590fbba

Author: Valeriya <lera.egorova-2000@yandex.ru>

Date: Fri Dec 25 12:25:55 2020 +0300

header PayRow added by Lera

**Добавление файла PayRow.h в ветку featurePay.**

1. commit ea6f21070b9b9030aa639e655a57607caa67b9aa

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Thu Dec 24 18:24:50 2020 +0300

header ClientInputScreen added

**Добавление файла ClientInputScreen.h в ветку featureClient.**

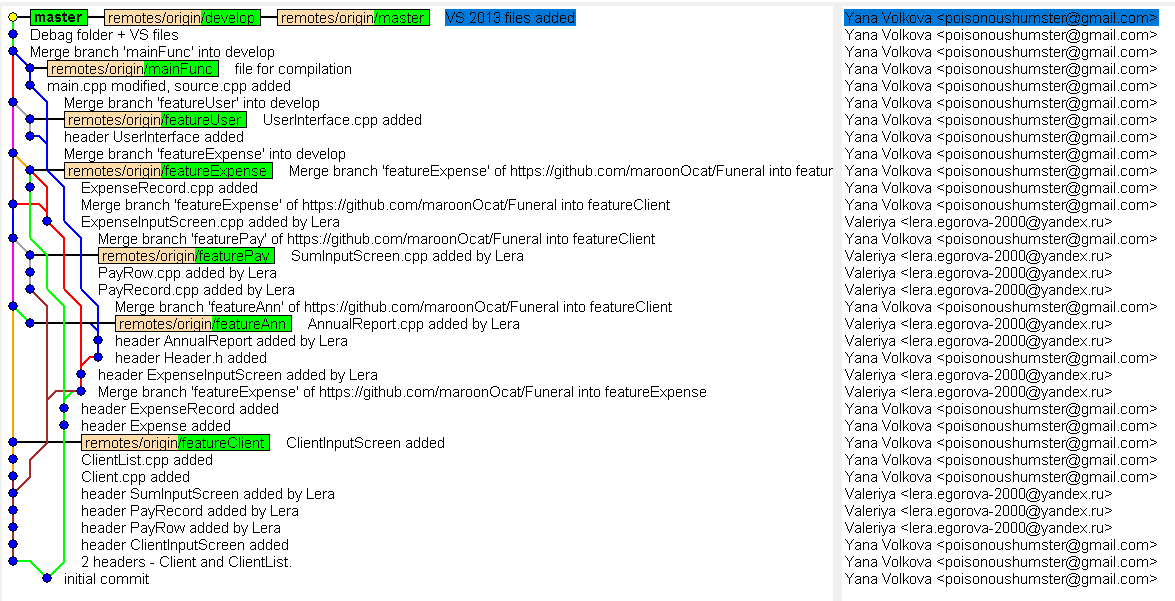
1. commit 39525ddd90050ab6e33a69e3ac24e15cd5459acb

Author: Yana Volkova <poisonoushumster@gmail.com>

Date: Thu Dec 24 17:59:50 2020 +0300

2 headers - Client and ClientList.

**Добавление файлов Client.h и ClientList.h в ветку featureClient.**

****

**Заключение**

Таким образом, была создана программа, которая могла бы использоваться работниками бюро ритуальных услуг. Программа была написана в среде разработки IDE Visual Studio 2013 Express на языке С++. В процессе работы применялась система контроля версий Git, и были созданы локальный репозиторий и удаленный репозиторий на Github.com.

В данной командной работе были реализованы все прецеденты, сбои в процессе работы программы не наблюдаются, как и зависания. Были использованы принципы раздельной компиляции и реализована отчистка памяти. Программа не содержит неиспользуемых переменных или избыточных алгоритмов. В Приложении к данному отчету находится 8 диаграмм, созданных с помощью программ WhiteStarUML и StarUML: диаграмма вариантов использования, 3 диаграммы действий, диаграмма классов и 3 диаграммы последовательностей. Цель, поставленная перед началом работы, была достигнута.

**Приложение**



Рис. 1. Диаграмма вариантов использования

****

Рис. 2. Диаграмма действий «Ввести услугу/товар»



Рис. 3. Диаграмма действий «Ввести клиента»



Рис. 4. Диаграмма действий «Вывести расходы»

****

Рис. 5. Диаграмма классов

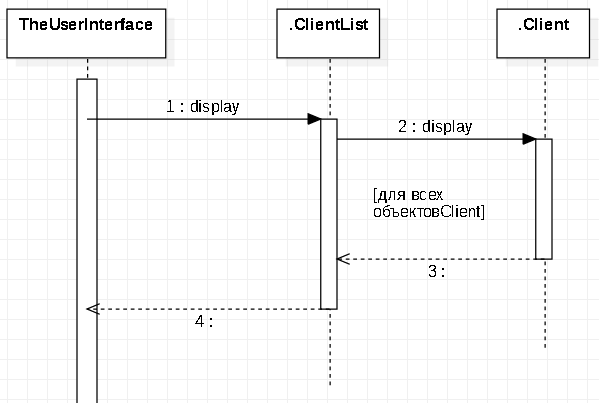


Рис. 6. Диаграмма последовательностей для варианта использования “Вывод списка клиентов”

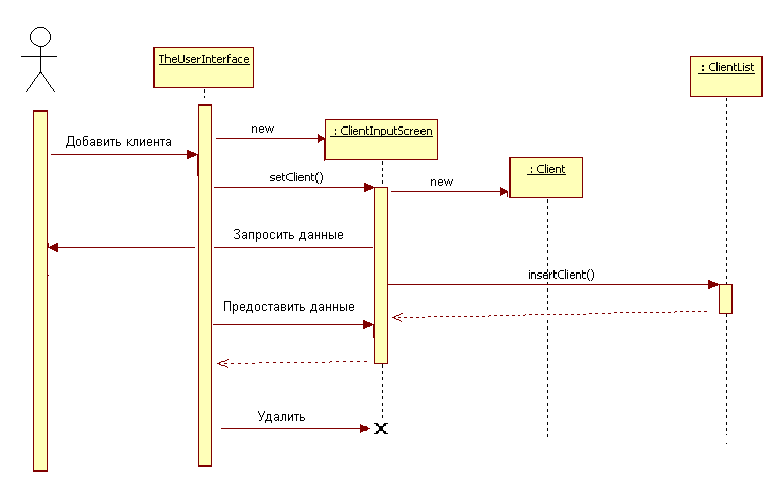


Рис. 7. Диаграмма последовательностей для варианта использования “Добавить нового клиента”

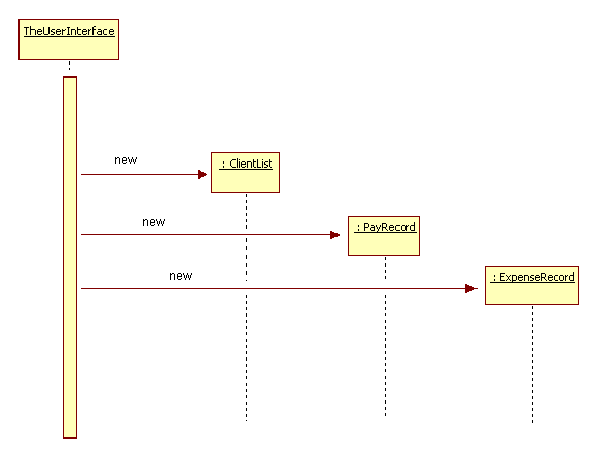


Рис. 8. Диаграмма последовательностей для варианта использования “Начать программу”