ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

старший преподаватель		Шумова Е.О.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

ОПИСАНИЕ КЛАССОВ И ПОРОЖДЕНИЕ ОБЪЕКТОВ

по курсу: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4033		Х.В. Сидиропуло
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

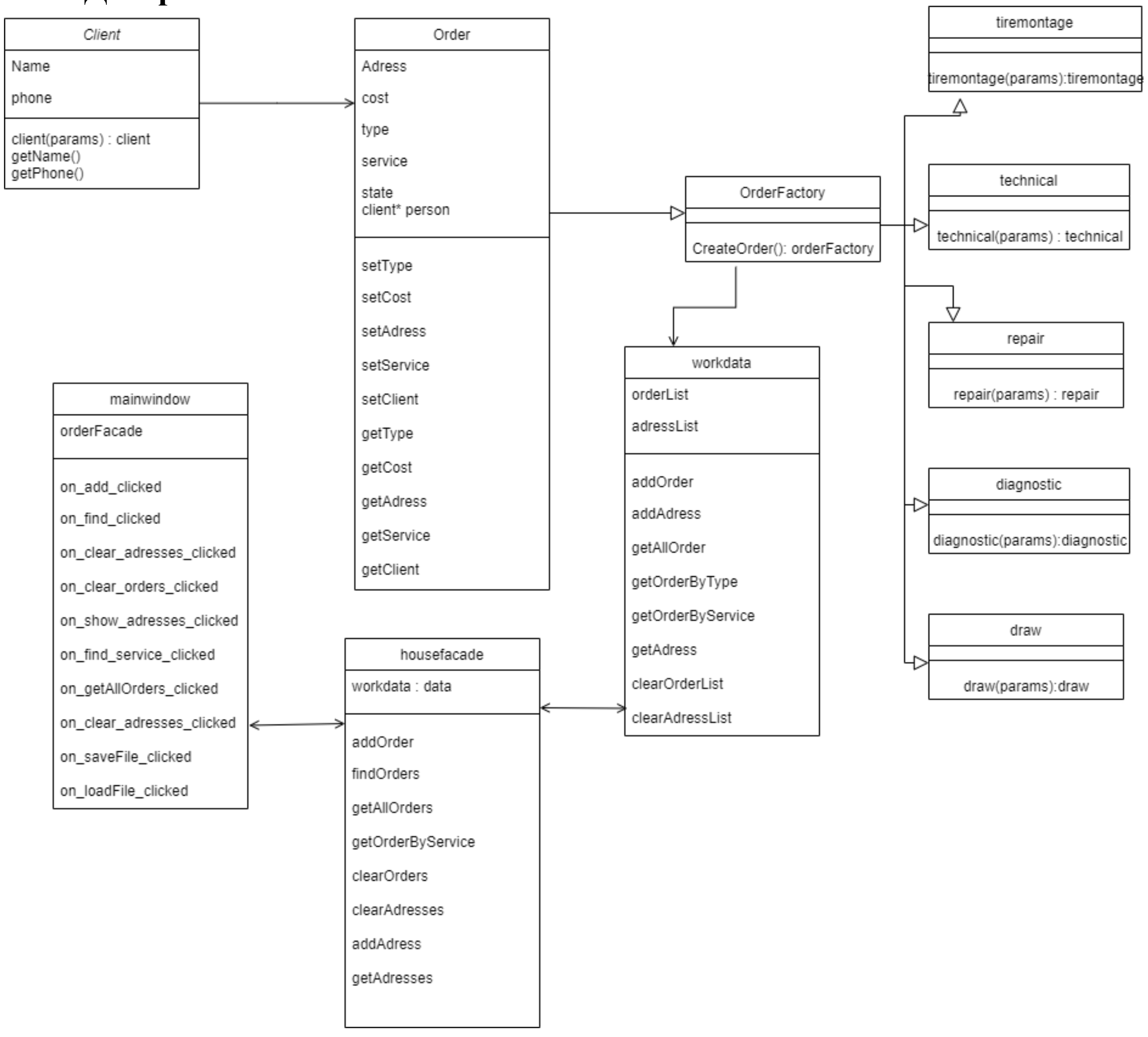
Лабораторная работа №9
«Описание классов и порождение объектов»
Вариант №6

Цель работы:

Спроектировать и реализовать информационную систему по указанной тематике - ремонтная мастерская.

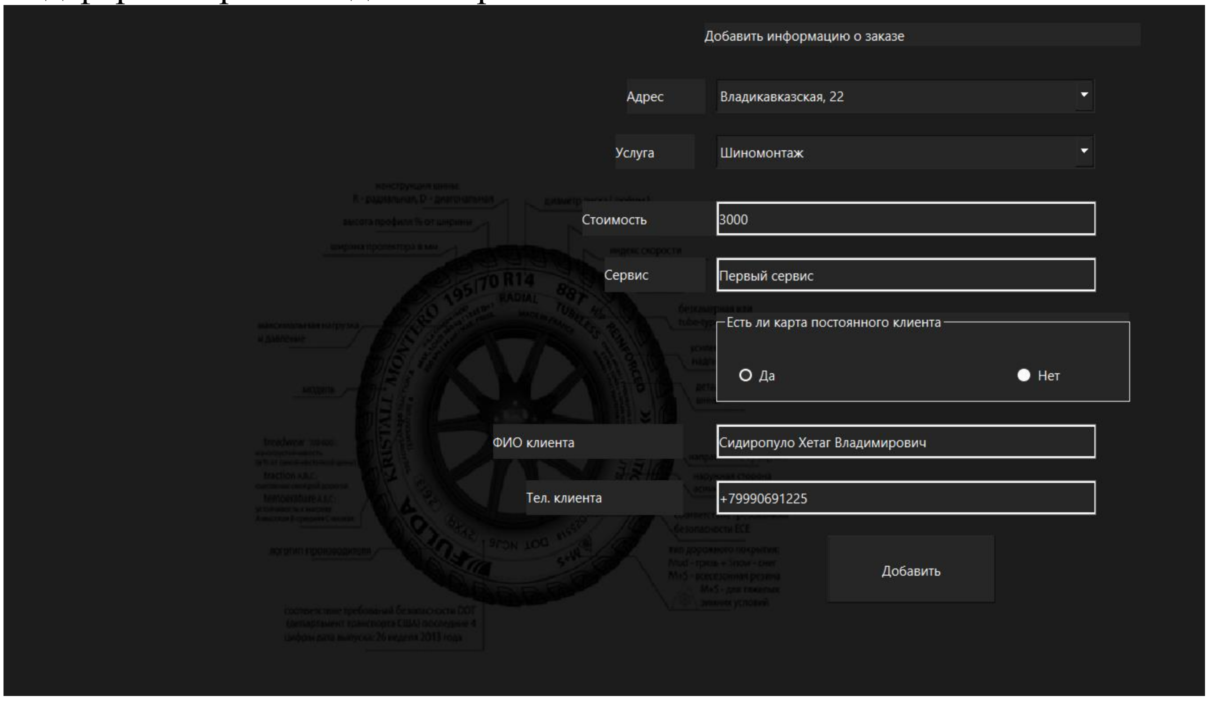
Построить диаграмму классов в нотации UML

Диаграмма классов



Вид исходной формы

Вид формы в режиме дизайнера



Добавить информацию о заказе

Адрес: Владикавказская, 22

Услуга: Шиномонтаж

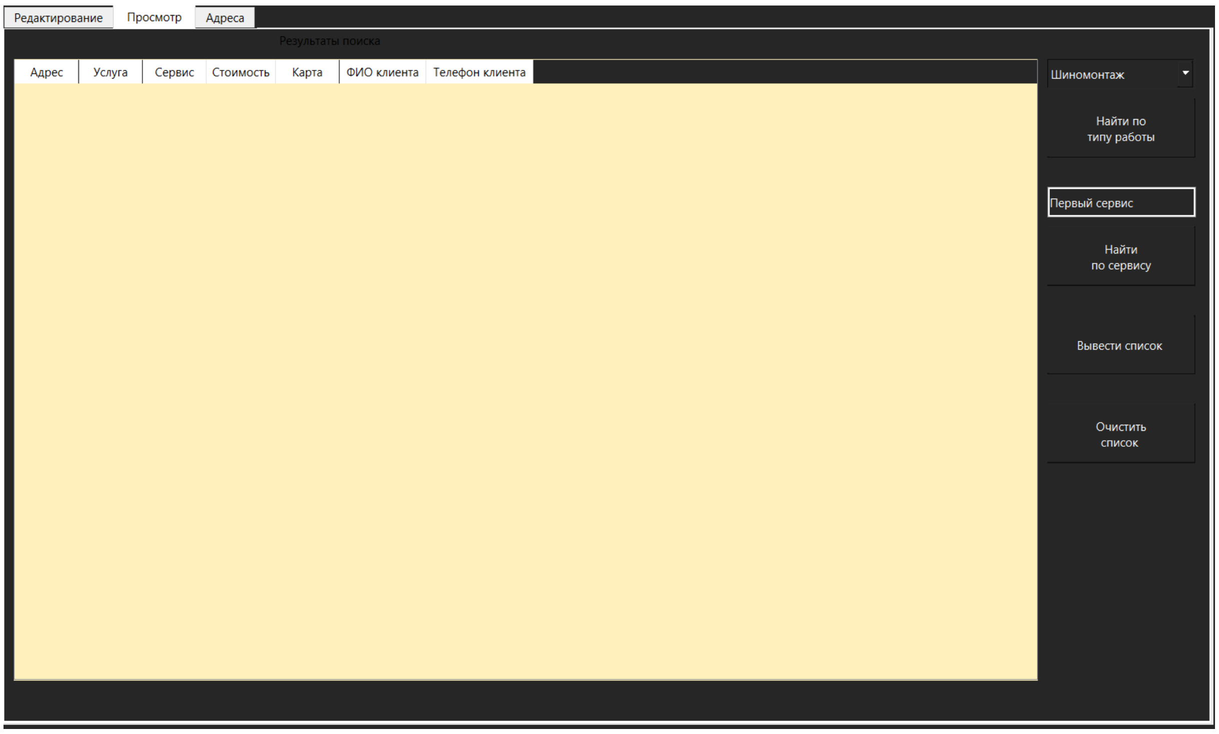
Стоимость: 3000

Сервис: Первый сервис

Есть ли карта постоянного клиента: Да [✓ Нет]

ФИО клиента: Сидиропуло Хетаг Владимирович

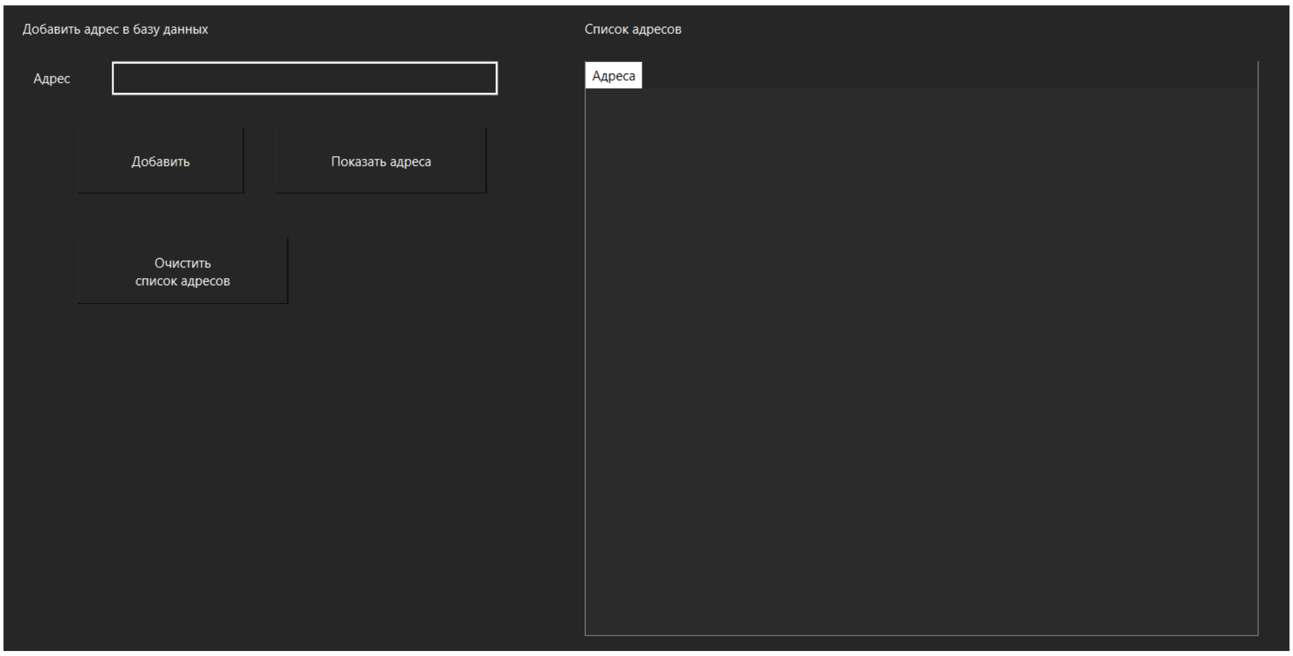
Тел. клиента: +79990691225



Адрес	Услуга	Сервис	Стоимость	Карта	ФИО клиента	Телефон клиента

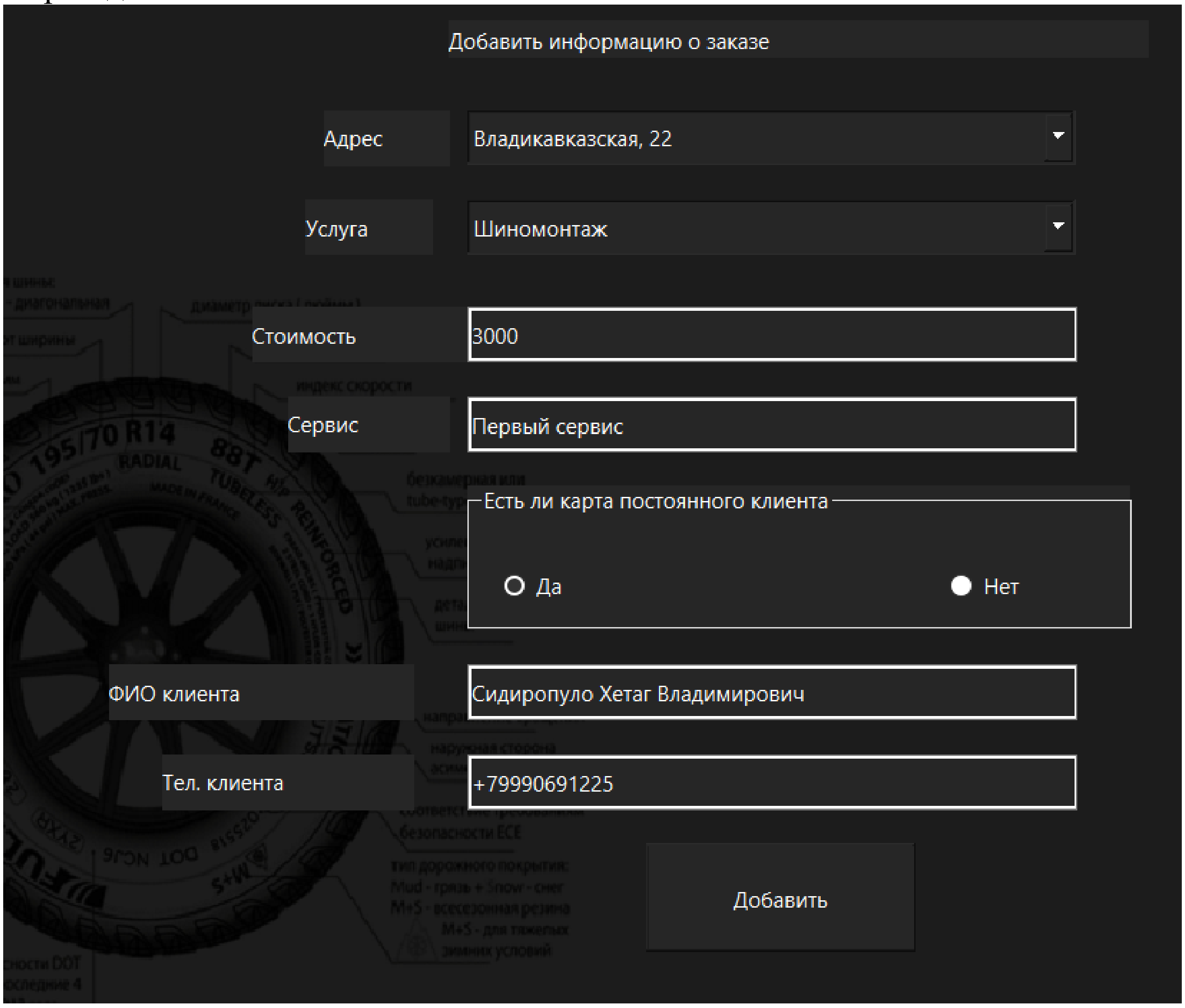
Шиномонтаж

Первый сервис


Добавить адрес в базу данных
Адрес
Добавить
Показать адреса
Очистить список адресов
Список адресов
Адреса

Демонстрация работы:

Форма добавления клиента:



Добавить информацию о заказе

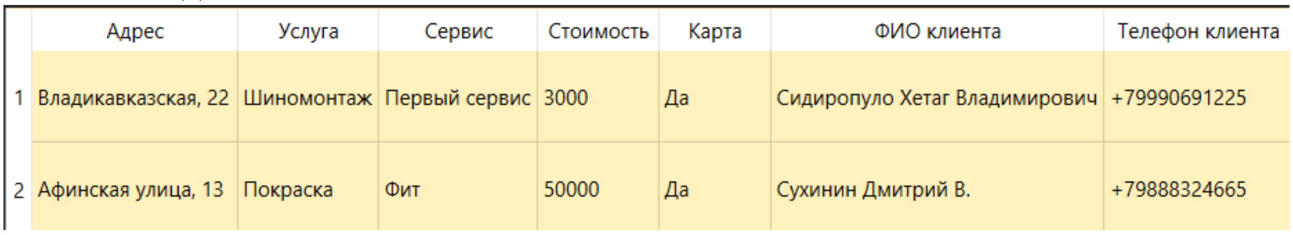
Адрес	Владикавказская, 22
Услуга	Шиномонтаж
Стоимость	3000
Сервис	Первый сервис

Есть ли карта постоянного клиента

○ Да ● Нет

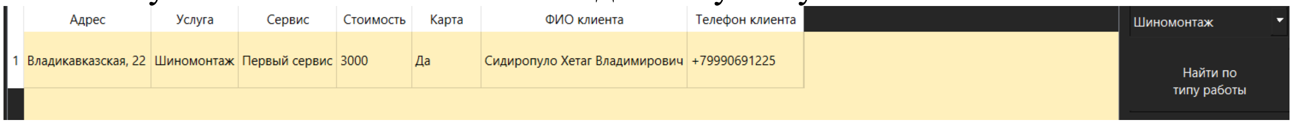
ФИО клиента	Сидиропуло Хетаг Владимирович
Тел. клиента	+79990691225

Вывод всего списка клиентов:



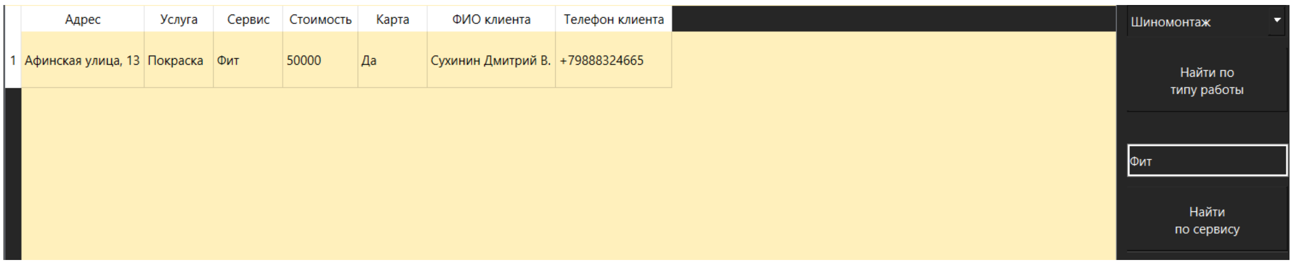
	Адрес	Услуга	Сервис	Стоимость	Карта	ФИО клиента	Телефон клиента
1	Владикавказская, 22	Шиномонтаж	Первый сервис	3000	Да	Сидиропуло Хетаг Владимирович	+79990691225
2	Афинская улица, 13	Покраска	Фит	50000	Да	Сухинин Дмитрий В.	+79888324665

Результат поиска клиента по заданному типу:



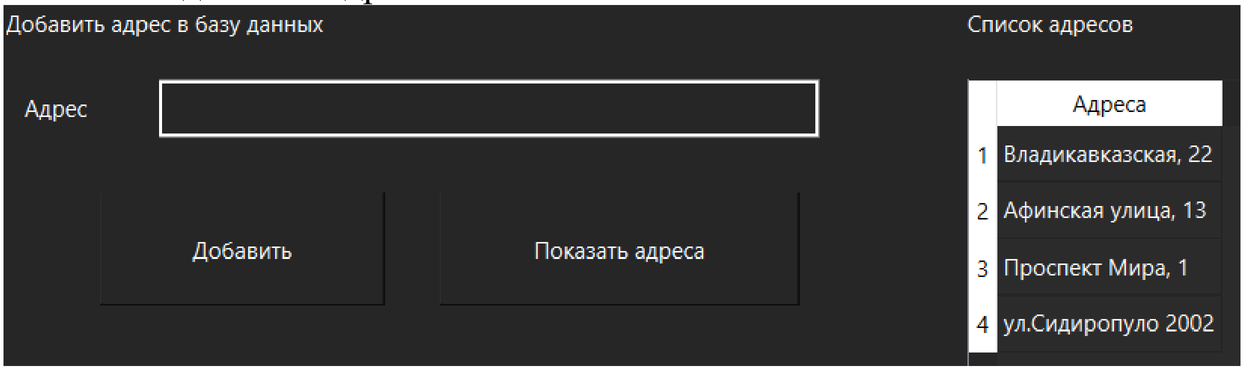
	Адрес	Услуга	Сервис	Стоимость	Карта	ФИО клиента	Телефон клиента
1	Владикавказская, 22	Шиномонтаж	Первый сервис	3000	Да	Сидиропуло Хетаг Владимирович	+79990691225

Результат поиска клиента по сервису:



	Адрес	Услуга	Сервис	Стоимость	Карта	ФИО клиента	Телефон клиента
1	Афинская улица, 13	Покраска	Фит	50000	Да	Сухинин Дмитрий В.	+79888324665

Вывод списка адресов:



	Адреса
1	Владикавказская, 22
2	Афинская улица, 13
3	Проспект Мира, 1
4	ул.Сидиропуло 2002

Листинг программы:

main.cpp
```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Client.h
```
#ifndef CLIENT_H
#define CLIENT_H

#include <QObject>

class client : public QObject
{
    Q_OBJECT
    QString name, phone;
public:
    client();
    client( QString name, QString phone);
    QString getName();
    QString getPhone();
    bool operator=(client &p1);
    ~client();
};

#endif // CLIENT_H
```

Diagnostic.h
```
#ifndef DIAGNOSTIC_H
#define DIAGNOSTIC_H

#include <QObject>
#include <orderfactory.h>

class diagnost : public orderfactory
{
    Q_OBJECT
public:
    diagnost();
    diagnost(QString adress, int cost, QString service, int state, client*
person);
};

#endif // DIAGNOSTIC_H
```

Draw.h
```
#ifndef DRAW_H
```

```
#define DRAW_H

#include <QObject>
#include <orderfactory.h>

class draw : public orderfactory
{
    Q_OBJECT
public:
    draw();
    draw(QString adress, int cost, QString service, int state, client* person);
};

#endif // DRAW_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <orderfacade.h>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

#define COLUMN_CNT 7

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void updateAdresses();

private slots:
    void on_add_clicked();

    void on_find_clicked();

    void on_getAllWork_clicked();

    void on_saveFile_clicked();

    void on_loadFile_clicked();

    void on_find_dist_clicked();

    void on_add_adress_clicked();

    void on_show_adress_clicked();

    void on_clearOrders_clicked();

    void on_clearAdresses_clicked();

private:
    Ui::MainWindow *ui;
    orderfacade orderFacade;
};
#endif // MAINWINDOW_H
```

order.h
```
#ifndef ORDER_H
#define ORDER_H

#include <QObject>
#include <client.h>

class order : public QObject
{
    Q_OBJECT
    QString adress;
    int cost;
    int type;
    int state; //0 - not sold, 1 - sold
    QString service;
    client* person;
public:
    order();
    order(QString adress, int cost, int type, QString service, int state,
client* person);

    void setType(int type);
    void setCost(int cost);
    void setAdress(QString adress);
    void setState(int state);
    void setService(QString service);
    void setClient(client* person);

    int getType();
    QString getAdress();
    QString getService();
    int getState();
    int getCost();
    client* getClient();
};

#endif // ORDER_H
```

Orderfacade.h
```
#ifndef ORDERFACADE_H
#define ORDERFACADE_H

#include <workdata.h>

class orderfacade : public QObject
{
    Q_OBJECT
    workdata data;
    QString fileNameFlats, fileNameAdresses;
public:
    orderfacade();
    void addOrder(order* h);
    QVector<QVector<QString>> findOrders(int type);
    QVector<QVector<QString>> getAllOrders();
    QVector<QVector<QString>> getOrderByService(QString dist);
    void clearOrder();

    void addAdress(QString adr);
    QVector<QString> getAdresses();
    void clearAdresses();
    void saveDataInFile();
    void loadDataFromFile();
};

#endif // ORDERFACADE_H
```

orderfactory.h

```
#ifndef ORDERFACTORY_H
#define ORDERFACTORY_H

#include <QObject>
#include <order.h>

class orderfactory : public order
{
    Q_OBJECT
public:
    orderfactory();
    orderfactory* createOrder(order* h);
};

#endif // ORDERFACTORY_H
```

Repair.h

```
#ifndef REPAIR_H
#define REPAIR_H

#include <QObject>
#include <orderfactory.h>

class repair : public orderfactory
{
    Q_OBJECT
public:
    repair();
    repair(QString adress, int cost, QString service, int state, client*
person);
};

#endif // REPAIR_H
```

Technical.h

```
#ifndef TECHNICAL_H
#define TECHNICAL_H

#include <QObject>
#include <orderfactory.h>

class technic
        : public orderfactory
{
    Q_OBJECT
public:
    technic();
    technic(QString adress, int cost, QString service, int state, client*
person);
};

#endif // TECHNICAL_H
```

Tiremontage.h

```
#ifndef TIREMONTAGE_H
#define TIREMONTAGE_H

#include <QObject>
#include <orderfactory.h>

class tire : public orderfactory
{
    Q_OBJECT
public:
    tire();
    tire(QString adress, int cost, QString service, int state, client* person);
};

#endif // TIREMONTAGE_H
```

Workdata.h

```
#ifndef WORKDATA_H
#define WORKDATA_H

#include <QFile>
#include <QTextStream>
#include <vector>
#include <orderfactory.h>

class workdata : public QObject
{
    Q_OBJECT
    QList<orderfactory *> carList;
    QList<QString> adressList;
public:
    workdata();
    void addOrder(orderfactory* stud);
    void addAdress(QString adress);
    QVector<QVector<QString>> getAllOrders();
    QVector<QVector<QString>> getOrdersByType(int type);
    QVector<QVector<QString>> getOrdersByService(QString dist);
    QVector<QString> getAdresses();
    void clearOrderList();
    void clearAdressList();

    void saveInFile(QString flats, QString adresses);
    void loadFromFile(QString flats, QString adresses);
};

#endif // WORKDATA_H
```

Client.cpp

```
#include "client.h"

client::client()
{

}

client::~client() {}

client::client( QString name, QString phone)
{
```

```cpp
    this->name = name;
    this->phone = phone;
}

QString client::getName()
{
    return name;
}

QString client::getPhone()
{
    return phone;
}

bool client::operator=(client &p1)
{
    return (this->name == p1.name) && (this->phone == p1.phone);
}
```

Diagnostic.cpp

```cpp
#include "diagnostic.h"

diagnost::diagnost()
{
    this->setType(4);
}

diagnost::diagnost(QString adress, int cost, QString service, int state, client*
person)
{
    this->setType(4);
    this->setAdress(adress);
    this->setCost(cost);
    this->setService(service);
    this->setState(state);
    this->setClient(person);
}
```

Draw.cpp

```cpp
#include "draw.h"

draw::draw()
{
    this->setType(1);
}

draw::draw(QString adress, int cost, QString service, int state, client* person)
{
    this->setType(1);
    this->setAdress(adress);
    this->setCost(cost);
    this->setService(service);
    this->setState(state);
    this->setClient(person);
}
```

MainWindow.cpp

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    orderFacade.loadDataFromFile();

    updateAdresses();
    ui->cost->setText("3000");
    ui->service->setText("Первый сервис");
    ui->service_find->setText("Первый сервис");
    ui->yes_bought->setChecked(true);
    ui->client_name->setText("Сидиропуло Хетаг Владимирович");
    ui->client_phone->setText("+79990691225");
    for (int i = 0; i < COLUMN_CNT; i++)
        ui->tableOut->horizontalHeader()->setSectionResizeMode(i,
QHeaderView::ResizeToContents);
    ui->out_adress->horizontalHeader()->setSectionResizeMode(0,
QHeaderView::ResizeToContents);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::updateAdresses()
{
    QVector<QString> res = orderFacade.getAdresses();
    if (!res.size())
        ui->adressList->setEnabled(false);
   else
    {
        ui->adressList->clear();
        for (int i = 0; i < res.size(); i++)
           ui->adressList->addItem(res[i], i);
    }
}

void MainWindow::on_add_clicked()
{
    QString adress = ui->adressList->currentText(),
            service = ui->service->text(),
            fio = ui->client_name->text(),
            phone = ui->client_phone->text();
    int cost = ui->cost->text().toInt(),
        type = ui->type->currentIndex(),
        state;
    if (ui->yes_bought->isChecked())
        state = 1;
    else
    {
        state = 0;
        fio = "-";
        phone = "-";
    }
    client* cl = new client(fio, phone);
    order* h = new order(adress, cost, type, service, state, cl);
    orderFacade.addOrder(h);
    ui->cost->clear();
    ui->service->clear();
    ui->client_name->clear();
    ui->client_phone->clear();
```

```
}


void MainWindow::on_find_clicked()
{
    int type = ui->type_find->currentIndex();
    QVector<QVector<QString>> res = orderFacade.findOrders(type);
    if (!res.size())
    {
        ui->tableOut->setRowCount(1);
        ui->tableOut->setItem(0,0, new QTableWidgetItem("Ничего не найдено"));
    }
    else
    {
        ui->tableOut->setRowCount(res.size());
        for (int i = 0; i < res.size(); i++)
            for (int j = 0; j < COLUMN_CNT; j++)
                ui->tableOut->setItem(i, j, new QTableWidgetItem(res[i][j]));
    }
}

void MainWindow::on_getAllWork_clicked()
{
    QVector<QVector<QString>> res = orderFacade.getAllOrders();
    if (!res.size())
    {
        ui->tableOut->clearContents();
        ui->tableOut->setRowCount(1);
        ui->tableOut->setItem(0,0, new QTableWidgetItem("Ничего не найдено"));
    }
    else
    {
        ui->tableOut->setRowCount(res.size());
        for (int i = 0; i < res.size(); i++)
            for (int j = 0; j < COLUMN_CNT; j++)
                ui->tableOut->setItem(i, j, new QTableWidgetItem(res[i][j]));
    }
}


void MainWindow::on_saveFile_clicked()
{
    orderFacade.saveDataInFile();
}


void MainWindow::on_loadFile_clicked()
{
    orderFacade.loadDataFromFile();
}


void MainWindow::on_find_service_clicked()
{
    QString service = ui->service_find->text();
    QVector<QVector<QString>> res = orderFacade.getOrderByService(service);
    if (!res.size())
    {
        ui->tableOut->clearContents();
        ui->tableOut->setRowCount(1);
        ui->tableOut->setItem(0,0, new QTableWidgetItem("Ничего не найдено"));
    }
    else
    {
        ui->tableOut->setRowCount(res.size());
        for (int i = 0; i < res.size(); i++)
            for (int j = 0; j < COLUMN_CNT; j++)
                ui->tableOut->setItem(i, j, new QTableWidgetItem(res[i][j]));
    }
}


void MainWindow::on_add_adress_clicked()
{
    QString adr = ui->new_adress->text();
    orderFacade.addAdress(adr);
    ui->new_adress->clear();
    updateAdresses();
}


void MainWindow::on_show_adress_clicked()
{
    QVector<QString> res = orderFacade.getAdresses();
    if (!res.size())
    {
        ui->out_adress->clearContents();
        ui->out_adress->setRowCount(1);
        ui->out_adress->setItem(0,0, new QTableWidgetItem("Ничего не найдено"));
    }
    else
    {
        ui->out_adress->setRowCount(res.size());
        for (int i = 0; i < res.size(); i++)
           ui->out_adress->setItem(i, 0, new QTableWidgetItem(res[i]));
    }
}
void MainWindow::on_clearCars_clicked()
{
    orderFacade.clearOrder();
}


void MainWindow::on_clearAdresses_clicked()
{
    orderFacade.clearAdresses();
}
```

Order.cpp

```
#include "order.h"

order::order()
{

}

order::order(QString adress, int cost, int type, QString service, int state,
client* person)
{
    this->adress = adress;
    this->cost = cost;
    this->type = type;
    this->state = state;
    this->service = service;
    this->person = person;
}
```

```
void order::setType(int type)
{
    this->type = type;
}

void order::setCost(int cost)
{
    this->cost = cost;
}

void order::setAdress(QString adress)
{
    this->adress = adress;
}

void order::setService(QString service)
{
    this->service = service;
}

void order::setState(int state)
{
    this->state = state;
}

void order::setClient(client* person)
{
    this->person = person;
}

int order::getType()
{
    return type;
}

QString order::getAdress()
{
    return adress;
}

QString order::getService()
{
    return service;
}

int order::getCost()
{
    return cost;
}

int order::getState()
{
    return state;
}

client* order::getClient()
{
    return person;
}
```

Orderfacade.cpp

```
#include "orderfacade.h"
```

```
#include "orderfactory.h"

orderfacade::orderfacade()
{
    fileNameFlats = "../lab9/data.db";
    fileNameAdresses = "../lab9/adresses.db";
}

void orderfacade::addOrder(order* h)
{
    orderfactory* car;
    car = car->createOrder(h);
    data.addOrder(car);
}

QVector<QVector<QString>> orderfacade::findOrders(int type)
{
    return data.getOrdersByType(type);
}

QVector<QVector<QString>> orderfacade::getAllOrders()
{
    return data.getAllOrders();
}

QVector<QVector<QString>> orderfacade::getOrderByService(QString dist)
{
    return data.getOrdersByService(dist);
}

 void orderfacade::saveDataInFile()
 {
     data.saveInFile(fileNameFlats, fileNameAdresses);
 }

 void orderfacade::loadDataFromFile()
 {
     data.loadFromFile(fileNameFlats, fileNameAdresses);
 }

 void orderfacade::addAdress(QString adr)
 {
     data.addAdress(adr);
 }

 QVector<QString> orderfacade::getAdresses()
 {
     return data.getAdresses();
 }

 void orderfacade::clearAdresses()
 {
     data.clearAdressList();
 }

 void orderfacade::clearOrder()
 {
     data.clearOrderList();


 }
```

Orderfactory.cpp

```
#include "orderfactory.h"
```

```
#include "tiremontage.h"
#include "draw.h"
#include "technical.h"
#include "repair.h"
#include "diagnostic.h"

orderfactory::orderfactory()
{
}

orderfactory* orderfactory::createOrder(order* h)
{
    orderfactory* order;
    switch (h->getType())
    {
    case 0:
        order = new tire(h->getAdress(), h->getCost(), h->getService(), h-
>getState(), h->getClient());
        break;
    case 1:
        order = new draw(h->getAdress(), h->getCost(), h->getService(), h-
>getState(), h->getClient());
        break;
    case 2:
        order = new technic(h->getAdress(), h->getCost(), h->getService(), h-
>getState(), h->getClient());
        break;
    case 3:
        order = new repair(h->getAdress(), h->getCost(), h->getService(), h-
>getState(), h->getClient());
        break;
    case 4:
        order = new diagnost(h->getAdress(), h->getCost(), h->getService(), h-
>getState(), h->getClient());
        break;
    }
    return order;
}
```

Repair.cpp

```
#include "repair.h"

repair::repair()
{
    this->setType(3);
}

repair::repair(QString adress, int cost, QString service, int state, client*
person)
{
    this->setType(3);
    this->setAdress(adress);
    this->setCost(cost);
    this->setService(service);
    this->setState(state);
    this->setClient(person);
}
```

Technical.cpp

```
#include "technical.h"
```

```
technic::technic()
{
    this->setType(2);
}

technic::technic(QString adress, int cost, QString service, int state, client*
person)
{
    this->setType(2);
    this->setAdress(adress);
    this->setCost(cost);
    this->setService(service);
    this->setState(state);
    this->setClient(person);
}
```

Tiremontage.cpp

```
#include "tiremontage.h"

tire::tire()
{
    this->setType(0);
}

tire::tire(QString adress, int cost, QString service, int state, client* person)
{
    this->setType(0);
    this->setAdress(adress);
    this->setCost(cost);
    this->setService(service);
    this->setState(state);
    this->setClient(person);
}
```

Workdata.cpp

```
#include "workdata.h"
#include <QDebug>

workdata::workdata()
{

}
void workdata::addOrder(orderfactory* stud)
{
    OrderList.push_back(stud);
}

QVector<QVector<QString>> workdata::getAllOrders()
{
    QVector<QVector<QString>> res;
    QString tmp;
    for (int i = 0; i < OrderList.size(); i++)
    {
        QVector<QString> Orderinfo;
        Orderinfo.push_back(OrderList[i]->getAdress());
        switch (OrderList[i]->getType())
        {
        case 0:
            Orderinfo.push_back("Шиномонтаж");
            break;
        case 1:
            Orderinfo.push_back("Покраска");
            break;
        case 2:
            Orderinfo.push_back("Ремонт");
            break;
        case 3:
            Orderinfo.push_back("ТО");
            break;
        case 4:
            Orderinfo.push_back("Диагностика");
            break;
        }
        Orderinfo.push_back(OrderList[i]->getService());
        Orderinfo.push_back(tmp.setNum(OrderList[i]->getCost()));
        OrderList[i]->getState() ? Orderinfo.push_back("Да") :
Orderinfo.push_back("Нет");
        Orderinfo.push_back(OrderList[i]->getClient()->getName());
        Orderinfo.push_back(OrderList[i]->getClient()->getPhone());
        res.push_back(Orderinfo);
    }
    return res;
}

QVector<QVector<QString>> workdata::getOrdersByType(int type)
{
    QVector<QVector<QString>> res;
    QString tmp;
    int prevind = 0;
    for (int i = 0; i < OrderList.size(); i++)
    {
        if (OrderList[i]->getType() == type)
        {
            QVector<QString> Orderinfo;
            Orderinfo.push_back(OrderList[i]->getAdress());
            switch (OrderList[i]->getType())
            {
            case 0:
                Orderinfo.push_back("Шиномонтаж");
                break;
            case 1:
                Orderinfo.push_back("Покраска");
                break;
            case 2:
                Orderinfo.push_back("Ремонт");
                break;
            case 3:
                Orderinfo.push_back("ТО");
                break;
            case 4:
                Orderinfo.push_back("Диагностика");
                break;
            }
            Orderinfo.push_back(OrderList[i]->getService());
            Orderinfo.push_back(tmp.setNum(OrderList[i]->getCost()));
            OrderList[i]->getState() ? Orderinfo.push_back("Да") :
Orderinfo.push_back("Нет");
            Orderinfo.push_back(OrderList[i]->getClient()->getName());
            Orderinfo.push_back(OrderList[i]->getClient()->getPhone());
            res.push_back(Orderinfo);
            prevind++;
        }
    }
    return res;
}
```

```cpp
QVector<QVector<QString>> workdata::getOrdersByService(QString dist)
{
    QVector<QVector<QString>> res;
    QString tmp;
    int prevind = 0;
    for (int i = 0; i < OrderList.size(); i++)
    {
        if (OrderList[i]->getService() == dist)
        {
            QVector<QString> Orderinfo;
            Orderinfo.push_back(OrderList[i]->getAdress());
            switch (OrderList[i]->getType())
            {
            case 0:
                Orderinfo.push_back("Шиномонтаж");
                break;
            case 1:
                Orderinfo.push_back("Покраска");
                break;
            case 2:
                Orderinfo.push_back("Ремонт");
                break;
            case 3:
                Orderinfo.push_back("ТО");
                break;
            case 4:
                Orderinfo.push_back("Диагностика");
                break;
            }
            Orderinfo.push_back(OrderList[i]->getService());
            Orderinfo.push_back(tmp.setNum(OrderList[i]->getCost()));
            OrderList[i]->getState() ? Orderinfo.push_back("Да") :
Orderinfo.push_back("Нет");
            Orderinfo.push_back(OrderList[i]->getClient()->getName());
            Orderinfo.push_back(OrderList[i]->getClient()->getPhone());
            res.push_back(Orderinfo);
            prevind++;
        }
    }
    return res;
}

void workdata::saveInFile(QString flats, QString adresses)
{
    QFile file_flats(flats);
    if (file_flats.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream writeStream(&file_flats);
        for (int i = 0; i < OrderList.size(); i++)
        {
            writeStream << OrderList[i]->getAdress() + "\n" <<
                           OrderList[i]->getType() << "\n" <<
                           OrderList[i]->getService() + "\n"<<
                           OrderList[i]->getCost() << "\n" <<
                           OrderList[i]->getState() << "\n" <<
                           OrderList[i]->getClient()->getName() << "\n" <<
                           OrderList[i]->getClient()->getPhone() << "\n";
        }
        file_flats.close();
    }

    QFile file_adrs(adresses);
    if (file_adrs.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream writeStream(&file_adrs);
        for (int i = 0; i < adressList.size(); i++)
            writeStream << adressList[i] << "\n";
        file_adrs.close();
    }
}

void workdata::loadFromFile(QString flats, QString adresses)
{
    QFile file_flats(flats);
    if (file_flats.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        if (OrderList.size())
            OrderList.clear();
        QString adress, service, tmp, name, phone;
        int type, cost, state;
        do
        {
            adress = file_flats.readLine().trimmed();
            if (adress == "")
                break;
            tmp = file_flats.readLine();
            type = tmp.toInt();
            service = file_flats.readLine().trimmed();
            tmp = file_flats.readLine();
            cost = tmp.toInt();
            tmp = file_flats.readLine();
            state = tmp.toInt();
            name = file_flats.readLine().trimmed();
            phone = file_flats.readLine().trimmed();
            order* h = new order(adress, cost, type, service, state, new
client(name, phone));
            orderfactory* hf = hf->createOrder(h);
            addOrder(hf);

        } while (!tmp.isNull());
        file_flats.close();
    }

    QFile file_adrs(adresses);
    if (file_adrs.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        if (adressList.size())
            adressList.clear();
        QString adress;
        do
        {
            adress = file_adrs.readLine().trimmed();
            if (adress == "")
                break;
            adressList.push_back(adress);
        } while (!adress.isNull());
        file_adrs.close();
    }
}

void workdata::addAdress(QString adress)
{
    adressList.push_back(adress);
}

QVector<QString> workdata::getAdresses()
{
    QVector<QString> res;
    for (auto adr : adressList)
        res.push_back(adr);
    return res;
}

void workdata::clearOrderList()
{
    if (OrderList.size())
        OrderList.clear();
}

void workdata::clearAdressList()
{
    if (adressList.size())
        adressList.clear();
}
```

Вывод:

Получилась полностью рабочая разработанная система для заказов в ремонтной мастерской.