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»

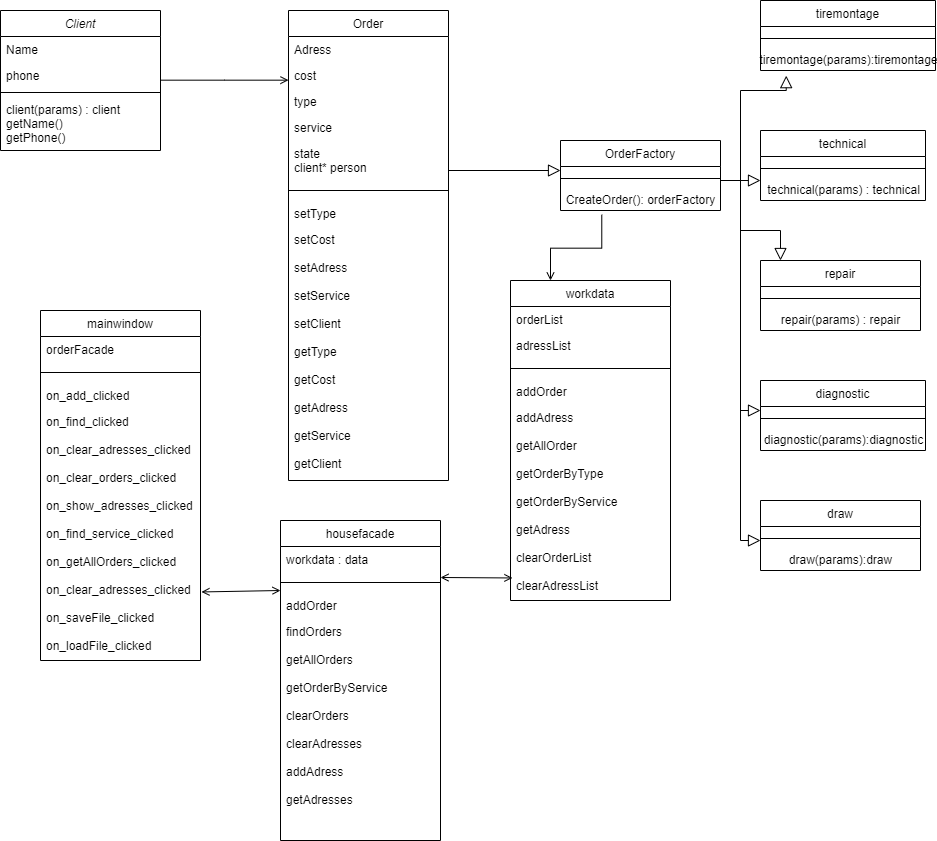
Вариант №6

**Цель работы:**

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

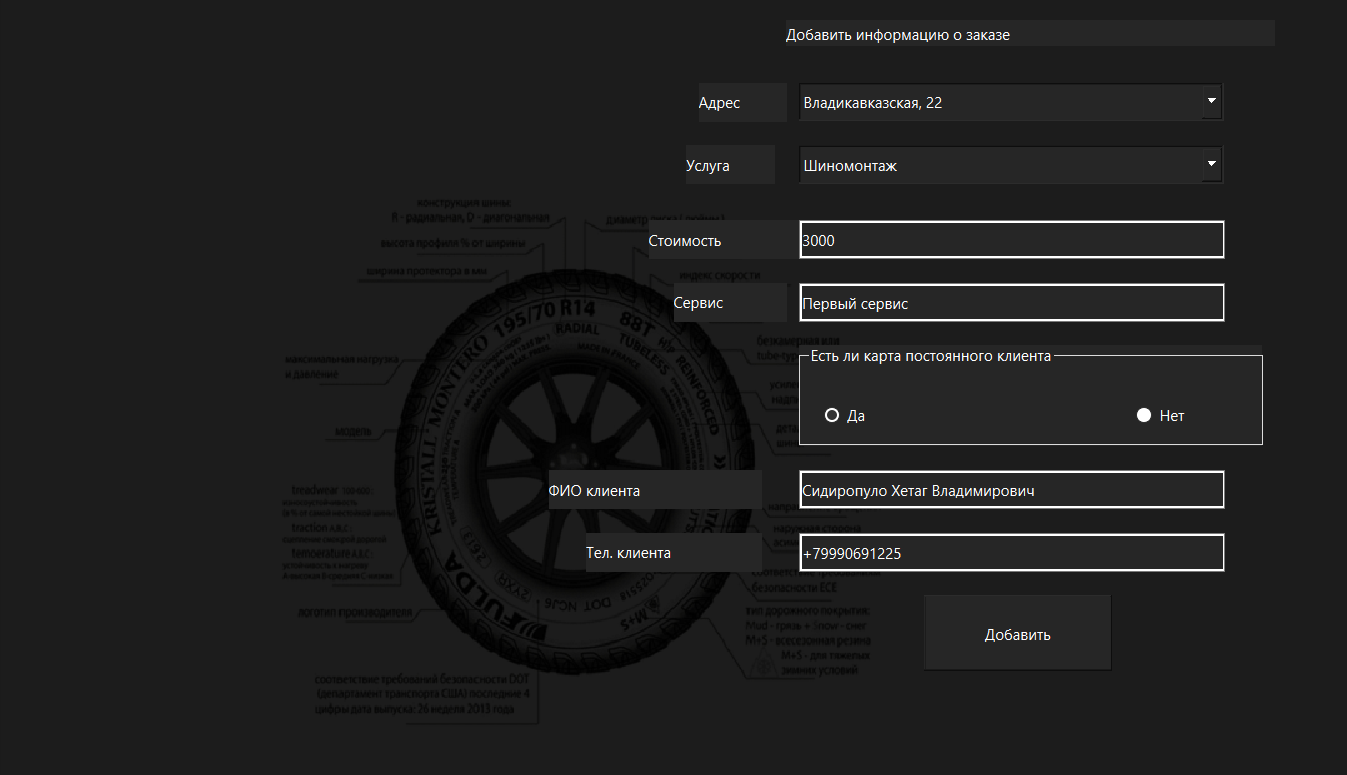
Построить диаграмму классов в нотации UML

**Диаграмма классов**

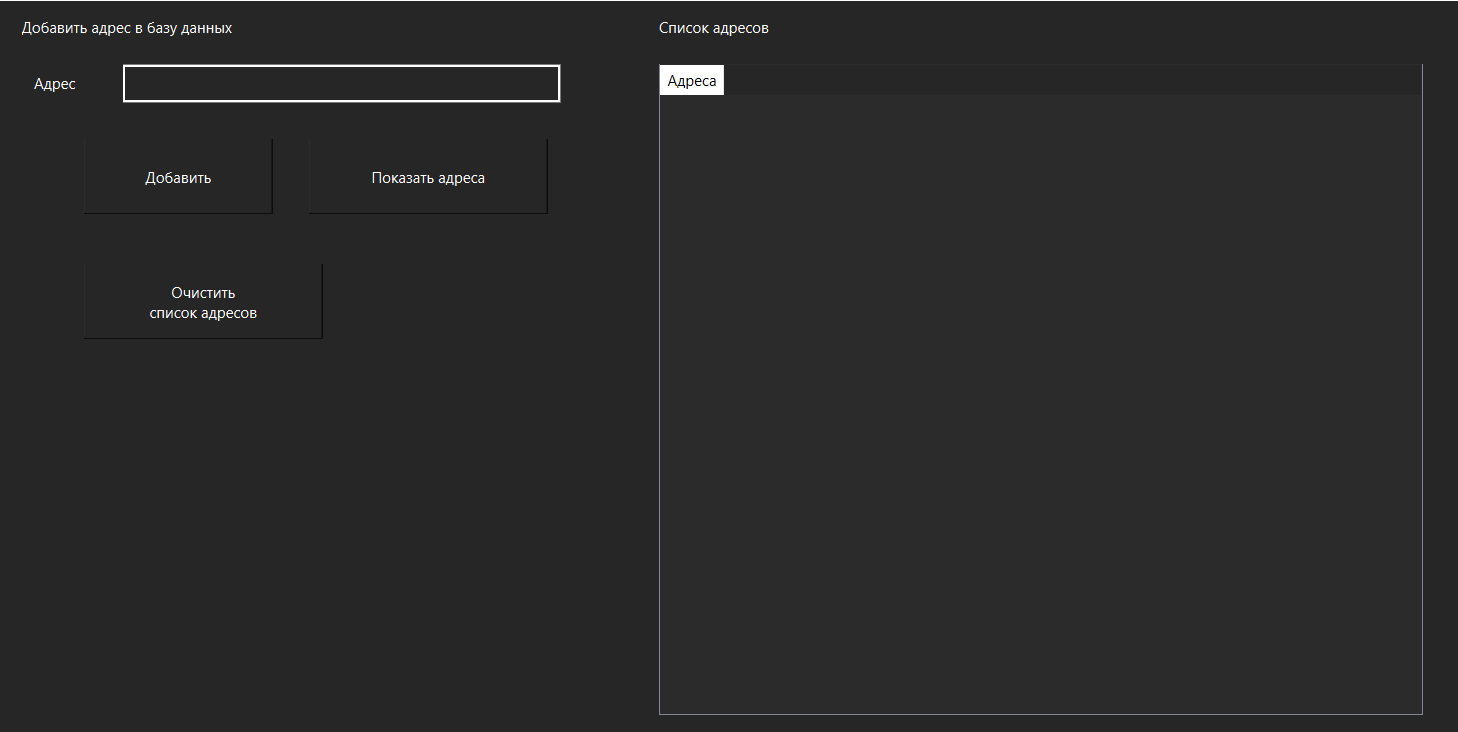
****

**Вид исходной формы**

Вид формы в режиме дизайнера

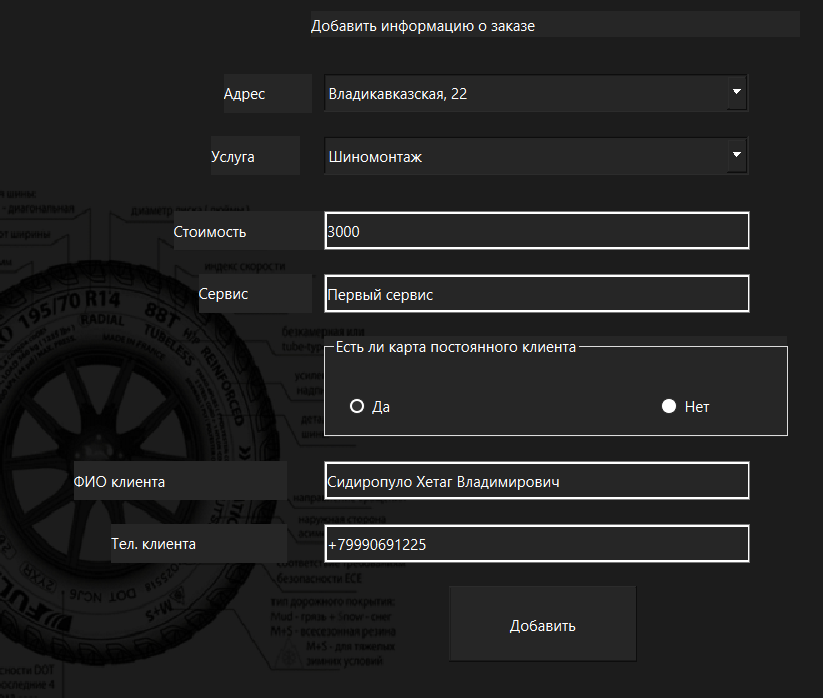




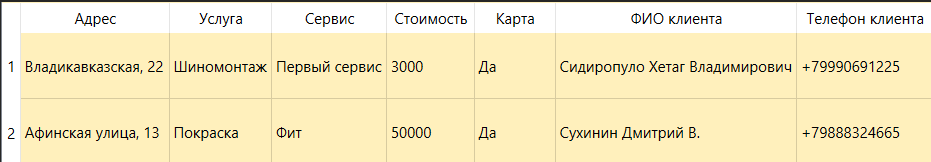


**Демонстрация работы:**

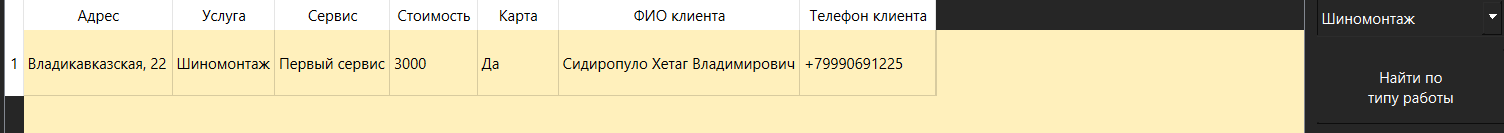
Форма добавления клиента:



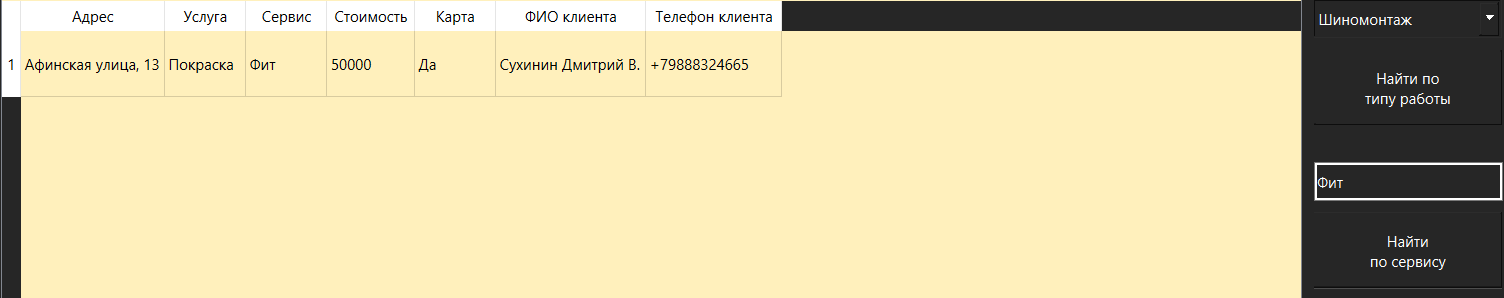
Вывод всего списка клиентов:

****

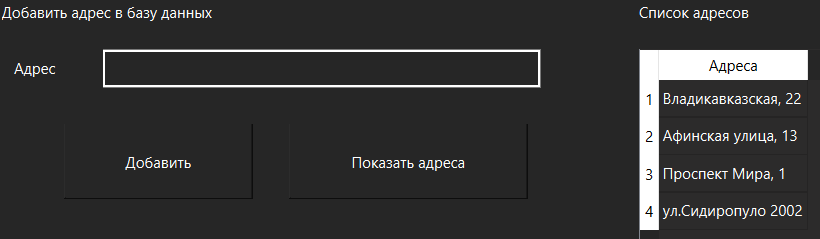
Результат поиска клиента по заданному типу:



Результат поиска клиента по сервису:



Вывод списка адресов:

****

**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)

{

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

#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

#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

#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;

}

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.