

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

старший преподаватель

должность, уч. степень, звание

подпись, дата

Шумова Е.О.

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

ОПИСАНИЕ КЛАССОВ И ПОРОЖДЕНИЕ ОБЪЕКТОВ

по курсу: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4033

подпись, дата

А.А.Фабрика

инициалы, фамилия

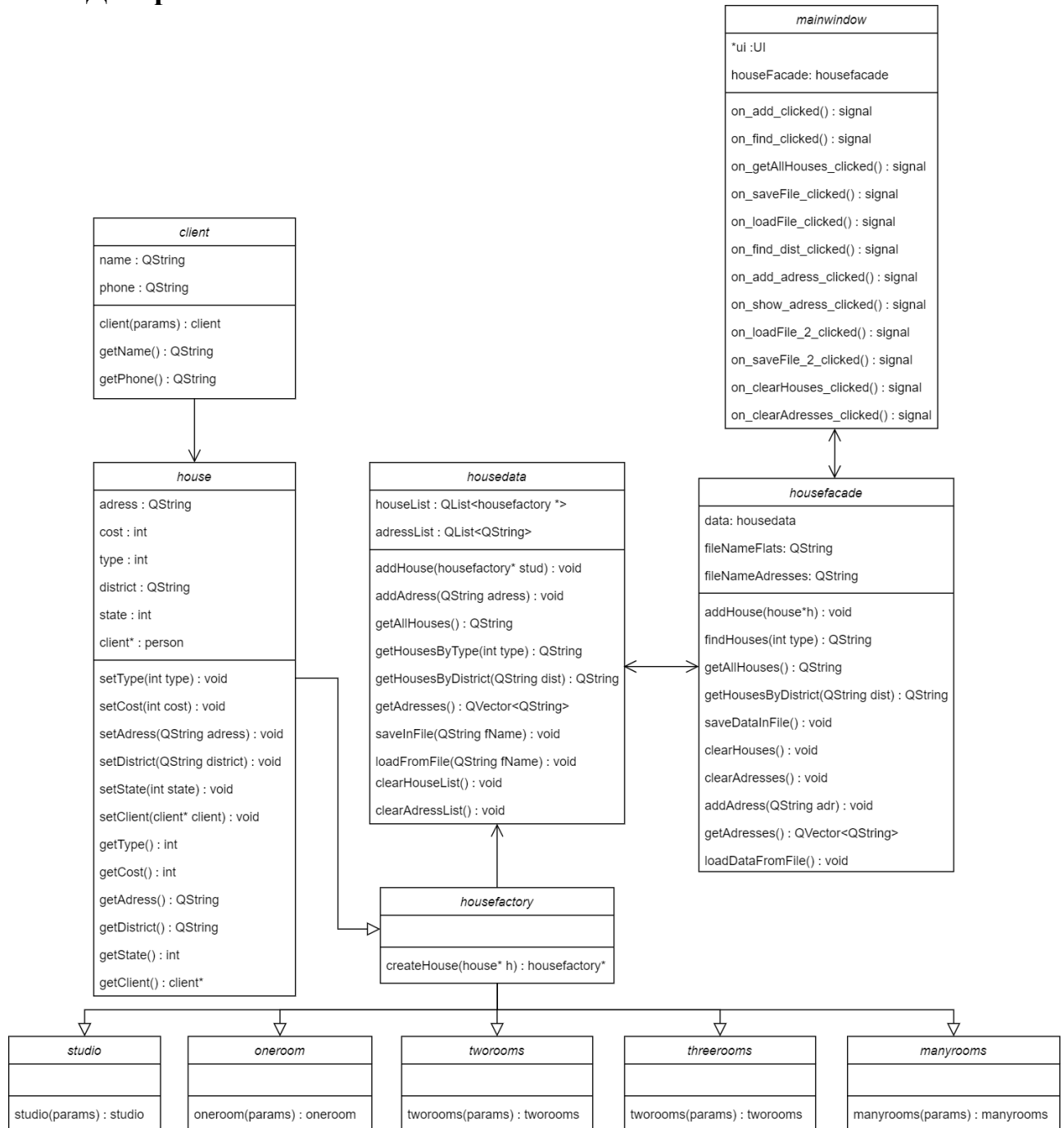
Санкт-Петербург 2022

Лабораторная работа №9
«Описание классов и порождение объектов»
Вариант №11

Цель работы:

Спроектировать и реализовать информационную систему по указанной тематике (Агентство недвижимости); построить диаграмму классов в нотации UML

Диаграмма классов



Вид исходной формы

Вид формы в режиме дизайнера

Редактирование

Просмотр

Адреса

Добавить информацию о квартире

Адрес

Тип

Студия

Стоимость

Район

Квартира куплена

☐ Да

☐ Нет

ФИО клиента

Тел. клиента

Добавить

Сохранить
данные в файл

Загрузить
данные из файла

Редактирование

Просмотр

Адреса

Результаты поиска

Адрес	Район	Тип	Стоимость	Куплена	ИО клиент	ефон клие
-------	-------	-----	-----------	---------	-----------	-----------

Студия

Найти по типу квартиры

Найти квартиру по району

Вывести список всех квартир

Очистить список квартир

Редактирование

Просмотр

Адреса

Добавить адрес в базу данных

Адрес

Добавить

Показать адреса

Сохранить данные в файл





Загрузить данные из файла

Очистить список адресов

Список адресов

Адреса

Виджеты, использующиеся в форме:

▼ tabWidget	QTabWidget
▼  tab	QWidget
add	QPushButton
adressList	QComboBox
client_name	QLineEdit
client_phone	QLineEdit
cost	QLineEdit
district	QLineEdit
▼  is_bought	QGroupBox
no_bought	QRadioButton
yes_bought	QRadioButton
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
label_5	QLabel
label_7	QLabel
label_8	QLabel
loadFile	QPushButton
saveFile	QPushButton
type	QComboBox
▼  tab_2	QWidget
clearHouses	QPushButton
district_find	QLineEdit
find	QPushButton
find_dist	QPushButton
getAllHouses	QPushButton
label_6	QLabel
tableOut	QTableWidget
type_find	QComboBox
▼  tab_3	QWidget
add_adress	QPushButton
clearAdresses	QPushButton
label_11	QLabel
label_12	QLabel
label_13	QLabel
loadFile_2	QPushButton
new_adress	QLineEdit
out_adress	QTableWidget
saveFile_2	QPushButton
show_adress	QPushButton

- tab (QWidget) – вкладка для управления записями о квартирах
- client_name, client_phone, cost, district (QLineEdit) – поля для ввода имени покупателя, телефона покупателя, стоимости квартиры, района
- type, adressList (QComboBox) – выпадающие списки для выбора типа квартиры и доступного адреса
- add, loadFile, saveFile (QPushButton) – кнопки для добавления квартиры в БД, загрузки и сохранения данных в БД
- label – label_13 (QLabel) – подписи используемых форм

- is_bought (QGroupBox) – выбор состояния квартиры (продана или нет)
- tab2 (QWidget) – вкладка для вывода записей о квартирах
- typeFind (QComboBox) – выпадающие списки для выбора типа запрашиваемой квартиры
- district_find (QLineEdit) – поле для ввода района, в котором находится запрашиваемая квартира
- clearHouses, find, find_dist, getAllHouses (QPushButton) – кнопки для очистки списка квартир, поиска квартиры по типу и району, вывода всего списка квартир
- tableOut (QTableWidget) – таблица, в которую выводится список квартир
- tab3 (QWidget) – вкладка для редактирования доступных адресов
- add_adress, loadFile_2, saveFile_2, clearAdresses, show_adress (QPushButton) – кнопки для добавления доступного адреса в БД, загрузки и сохранения данных в БД, очистки списка адресов, вывода списка адресов
- out_adress (QTableWidget) – таблица, в которую выводится список адресов
- district_find (QLineEdit) – поле для ввода нового доступного адреса

Демонстрация работы:

Начальное состояние формы:

MainWindow

РедактированиеПросмотрАдреса

Добавить информацию о квартире

Адрес

Лабораторный проспект, 22

Тип

Студия

Стоимость

3000000

Район

Калининский район

Квартира куплена

☒ Да☐ Нет

ФИО клиента

Иванов Виталий Олегович

Тел. клиента

+79219876543

Добавить

Сохранить
данные в файл

Загрузить
данные из файла

Форма после нажатия на кнопку «Добавить»:

MainWindow

РедактированиеПросмотрАдреса

Добавить информацию о квартире

Адрес

Лабораторный проспект, 22

Тип

Студия

Стоимость

Район

Квартира куплена

☒ Да☐ Нет

ФИО клиента

Тел. клиента

Добавить

Сохранить
данные в файл

Загрузить
данные из файла

Вывод всего списка квартир:

MainWindow

Редактирование

Просмотр

Адреса

Результаты поиска

	Адрес	Район	Тип	Стоимость	Куплена	ФИО клиента	Телефон
1	Лабораторный проспект, 22	Студия	Калининский район	2500000	Нет	-	-
2	Лабораторный проспект, 24	Студия	Калининский район	3000000	Да	Иванов Иван Иванович	892
3	Садовая улица, 26Б	Две комнаты	Центральный	15000000	Нет	-	-
4	проспект Стачек, 10	Одна комната	Кировский	8500000	Да	Петров Петр Петрович	-
5	Лабораторный проспект, 24	Две комнаты	Калининский район	6000000	Да	Иванов Петр Петрович	88
6	Конная ул., 6	Три комнаты	-	20000000	Нет	-	-
7	Чернорецкий пер., 76	Более трех комнат	-	29000000	Да	Иванов Виталий Олегович	+7
8	Лабораторный проспект, 22	Студия	Калининский район	3000000	Да	Иванов Виталий Олегович	+7

Студия

Найти по типу квартиры

Калининский район

Найти квартиру по району

Вывести список всех квартир

Очистить список квартир

Результат поиска квартир по заданному типу:

MainWindow

Редактирование

Просмотр

Адреса

Результаты поиска

	Адрес	Район	Тип	Стоимость	Куплена	ФИО клиента	Телефон клиента
1	Садовая улица, 26Б	Две комнаты	Центральный	15000000	Нет	-	-
2	Лабораторный проспект, 24	Две комнаты	Калининский район	6000000	Да	Иванов Петр Петрович	8812789654

2-комнатная

Найти по типу квартиры

Калининский район

Найти квартиру по району

Вывести список всех квартир

Очистить список квартир

Добавление нового адреса:

MainWindow

— □ ×

Редактирование | Просмотр | Адреса

Добавить адрес в базу данных

Адрес

Добавить

Показать адреса

Сохранить данные в файл

Загрузить данные из файла

Очистить список адресов

Список адресов

Адреса

Вывод списка адресов:

MainWindow

— □ ×

Редактирование | Просмотр | Адреса

Добавить адрес в базу данных

Адрес

Добавить

Показать адреса

Сохранить данные в файл

Загрузить данные из файла

Очистить список адресов

Список адресов

	Адреса
1	Лабораторный проспект, 22
2	Лабораторный проспект, 24
3	Садовая улица, 26Б
4	проспект Стачек, 10
5	Дунайский проспект, 23
6	Лиговский пр., 83
7	Боровая ул., 24
8	ул. Марата, 48
9	Конная ул., 6
10	Чернорецкий пер., 76
11	Старорусская ул., 11
12	Садовая 2

Листинг программы:

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <housefacade.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 updateAddresses();

private slots:
    void on_add_clicked();

    void on_find_clicked();

    void on_getAllHouses_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_loadFile_2_clicked();

    void on_saveFile_2_clicked();

    void on_clearHouses_clicked();

    void on_clearAddresses_clicked();

private:
    Ui::MainWindow *ui;
    housefacade houseFacade;
};
#endif // MAINWINDOW_H
```

main.cpp:

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

```

        MainWindow w;
        w.show();
        return a.exec();
}

```

mainwindow.cpp:

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    houseFacade.loadDataFromFile();
    updateAddresses();
    ui->cost->setText("3000000");
    ui->district->setText("Калининский район");
    ui->district_find->setText("Калининский район");
    ui->yes_bought->setChecked(true);
    ui->client_name->setText("Иванов Виталий Олегович");
    ui->client_phone->setText("+79219876543");
    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::updateAddresses()
{
    QVector<QString> res = houseFacade.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 address = ui->adressList->currentText(),
        district = ui->district->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);
    house* h = new house(address, cost, type, district, state, cl);
    houseFacade.addHouse(h);
    ui->cost->clear();
    ui->district->clear();
    ui->client_name->clear();
    ui->client_phone->clear();
}

void MainWindow::on_find_clicked()
{
    int type = ui->type_find->currentIndex();
    QVector<QVector<QString>> res = houseFacade.findHouses(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_getAllHouses_clicked()
{
    QVector<QVector<QString>> res = houseFacade.getAllHouses();
    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()
{
    houseFacade.saveDataInFile();
}

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

void MainWindow::on_find_dist_clicked()
{
    QString district = ui->district_find->text();
    QVector<QVector<QString>> res = houseFacade.getHousesByDistrict(district);

```

```

        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_address_clicked()
{
    QString adr = ui->new_address->text();
    houseFacade.addAddress(adr);
    ui->new_address->clear();
    updateAddresses();
}

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

void MainWindow::on_loadFile_2_clicked()
{
    houseFacade.loadDataFromFile();
}

void MainWindow::on_saveFile_2_clicked()
{
    houseFacade.saveDataInFile();
}

void MainWindow::on_clearHouses_clicked()
{
    houseFacade.clearHouses();
}

void MainWindow::on_clearAdresses_clicked()
{
    houseFacade.clearAdresses();
}

```

```
}
```

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
```

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);
}
```

Данный класс хранит информацию о клиенте

house.h:

```
#ifndef HOUSE_H
#define HOUSE_H

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

```

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

    void setType(int type);
    void setCost(int cost);
    void setAddress(QString address);
    void setState(int state);
    void setDistrict(QString district);
    void setClient(client* person);

    int getType();
    QString getAddress();
    QString getDistrict();
    int getState();
    int getCost();
    client* getClient();
};

#endif // HOUSE_H

```

house.cpp:

```

#include "house.h"

house::house()
{

}

house::house(QString address, int cost, int type, QString district, int state,
client* person)
{
    this->address = address;
    this->cost = cost;
    this->type = type;
    this->state = state;
    this->district = district;
    this->person = person;
}

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

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

void house::setAddress(QString address)
{

```

```

        this->address = address;
    }

void house::setDistrict(QString district)
{
    this->district = district;
}

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

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

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

QString house::getAddress()
{
    return address;
}

QString house::getDistrict()
{
    return district;
}

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

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

client* house::getClient()
{
    return person;
}

```

Данный класс хранит данные об одной квартире

```

housedata.h:
#ifndef HOUSEDATA_H
#define HOUSEDATA_H

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

class housedata : public QObject
{
    Q_OBJECT

```



```

        QList<housefactory*> houseList;
        QList<QString> addressList;
public:
    housedata();
    void addHouse(housefactory* stud);
    void addAddress(QString address);
    QVector<QVector<QString>> getAllHouses();
    QVector<QVector<QString>> getHousesByType(int type);
    QVector<QVector<QString>> getHousesByDistrict(QString dist);
    QVector<QString> getAdresses();
    void clearHouseList();
    void clearAddressList();

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

#endif // HOUSEDATA_H

```

housedata.cpp

```

#include "housedata.h"
#include <QDebug>

housedata::housedata()
{

}

void housedata::addHouse(housefactory* stud)
{
    houseList.push_back(stud);
}

QVector<QVector<QString>> housedata::getAllHouses()
{
    QVector<QVector<QString>> res;
    QString tmp;
    for (int i = 0; i < houseList.size(); i++)
    {
        QVector<QString> houseinfo;
        houseinfo.push_back(houseList[i]->getAddress());
        switch (houseList[i]->getType())
        {
            case 0:
                houseinfo.push_back("Студия");
                break;
            case 1:
                houseinfo.push_back("Одна комната");
                break;
            case 2:
                houseinfo.push_back("Две комнаты");
                break;
            case 3:
                houseinfo.push_back("Три комнаты");
                break;
            case 4:
                houseinfo.push_back("Более трех комнат");
                break;
        }
        houseinfo.push_back(houseList[i]->getDistrict());
        houseinfo.push_back(tmp.setNum(houseList[i]->getCost()));
        houseList[i]->getState() ? houseinfo.push_back("Да") :
houseinfo.push_back("Нет");
        houseinfo.push_back(houseList[i]->getClient()->getName());
    }
}

```

```

        houseinfo.push_back(houseList[i]->getClient()->getPhone());
        res.push_back(houseinfo);
    }
    return res;
}

QVector<QVector<QString>> housedata::getHousesByType(int type)
{
    QVector<QVector<QString>> res;
    QString tmp;
    int prevind = 0;
    for (int i = 0; i < houseList.size(); i++)
    {
        if (houseList[i]->getType() == type)
        {
            QVector<QString> houseinfo;
            houseinfo.push_back(houseList[i]->getAddress());
            switch (houseList[i]->getType())
            {
                case 0:
                    houseinfo.push_back("Студия");
                    break;
                case 1:
                    houseinfo.push_back("Одна комната");
                    break;
                case 2:
                    houseinfo.push_back("Две комнаты");
                    break;
                case 3:
                    houseinfo.push_back("Три комнаты");
                    break;
                case 4:
                    houseinfo.push_back("Более трех комнат");
                    break;
            }
            houseinfo.push_back(houseList[i]->getDistrict());
            houseinfo.push_back(tmp.setNum(houseList[i]->getCost()));
            houseList[i]->getState() ? houseinfo.push_back("Да") :
houseinfo.push_back("Нет");
            houseinfo.push_back(houseList[i]->getClient()->getName());
            houseinfo.push_back(houseList[i]->getClient()->getPhone());
            res.push_back(houseinfo);
            prevind++;
        }
    }
    return res;
}

QVector<QVector<QString>> housedata::getHousesByDistrict(QString dist)
{
    QVector<QVector<QString>> res;
    QString tmp;
    int prevind = 0;
    for (int i = 0; i < houseList.size(); i++)
    {
        if (houseList[i]->getDistrict() == dist)
        {
            QVector<QString> houseinfo;
            houseinfo.push_back(houseList[i]->getAddress());
            switch (houseList[i]->getType())
            {
                case 0:
                    houseinfo.push_back("Студия");
                    break;

```

```

        case 1:
            houseinfo.push_back("Одна комната");
            break;
        case 2:
            houseinfo.push_back("Две комнаты");
            break;
        case 3:
            houseinfo.push_back("Три комнаты");
            break;
        case 4:
            houseinfo.push_back("Более трех комнат");
            break;
    }
    houseinfo.push_back(houseList[i]->getDistrict());
    houseinfo.push_back(tmp.setNum(houseList[i]->getCost()));
    houseList[i]->getState() ? houseinfo.push_back("Да") :
houseinfo.push_back("Нет");
    houseinfo.push_back(houseList[i]->getClient()->getName());
    houseinfo.push_back(houseList[i]->getClient()->getPhone());
    res.push_back(houseinfo);
    prevind++;
    }
    return res;
}

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

    QFile file_adrs(addresses);
    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 housedata::loadFromFile(QString flats, QString addresses)
{
    QFile file_flats(flats);
    if (file_flats.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        if (houseList.size())
            houseList.clear();
        QString adress, district, tmp, name, phone;
        int type, cost, state;
    }
}

```

```

do
{
    address = file_flats.readLine().trimmed();
    if (address == "")
        break;
    tmp = file_flats.readLine();
    type = tmp.toInt();
    district = 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();
    house* h = new house(address, cost, type, district, state, new
client(name, phone));
    housefactory* hf = hf->createHouse(h);
    addHouse(hf);

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

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

}

void housedata::addAddress(QString address)
{
    addressList.push_back(address);
}

QVector<QString> housedata::getAddresses()
{
    QVector<QString> res;
    for (auto adr : addressList)
        res.push_back(adr);
    return res;
}

void housedata::clearHouseList()
{
    if (houseList.size())
        houseList.clear();
}

void housedata::clearAddressList()
{
    if (addressList.size())
        addressList.clear();
}

```

Данный класс используется для хранения всех данных в приложении

housefacade.h:

```
#ifndef HOUSEFACADE_H
#define HOUSEFACADE_H

#include <housedata.h>

class housefacade : public QObject
{
    Q_OBJECT
    housedata data;
    QString fileNameFlats, fileNameAdresses;
public:
    housefacade ();
    void addHouse (house* h);
    QVector<QVector<QString>> findHouses (int type);
    QVector<QVector<QString>> getAllHouses ();
    QVector<QVector<QString>> getHousesByDistrict (QString dist);
    void clearHouses ();

    void addAddress (QString adr);
    QVector<QString> getAdresses ();
    void clearAdresses ();

    void saveDataInFile ();
    void loadDataFromFile ();
};

#endif // HOUSEFACADE_H
```

housefacade.cpp:

```
#include "housefacade.h"
#include "housefactory.h"

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

void housefacade::addHouse (house* h)
{
    housefactory* house;
    house = house->createHouse (h);
    data.addHouse (house);
}

QVector<QVector<QString>> housefacade::findHouses (int type)
{
    return data.getHousesByType (type);
}

QVector<QVector<QString>> housefacade::getAllHouses ()
{
    return data.getAllHouses ();
}

QVector<QVector<QString>> housefacade::getHousesByDistrict (QString dist)
{
    return data.getHousesByDistrict (dist);
}
```

```

void housefacade::saveDataInFile ()
{
    data.saveInFile(fileNameFlats, fileNameAddresses);
}

void housefacade::loadDataFromFile ()
{
    data.loadFromFile(fileNameFlats, fileNameAddresses);
}

void housefacade::addAddress (QString adr)
{
    data.addAddress(adr);
}

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

void housefacade::clearAdresses ()
{
    data.clearAddressList();
}

void housefacade::clearHouses ()
{
    data.clearHouseList();
}

```

Данный класс является посредником между классами обработки данных и клиентским кодом

housefactory.h:

```

#ifndef HOUSEFACTORY_H
#define HOUSEFACTORY_H

#include <QObject>
#include <house.h>

class housefactory : public house
{
    Q_OBJECT
public:
    housefactory();
    housefactory* createHouse(house* h);
};

#endif // HOUSEFACTORY_H

```

housefactory.cpp:

```

#include "housefactory.h"
#include "studio.h"
#include "oneroom.h"
#include "tworooms.h"
#include "threerooms.h"
#include "manyrooms.h"

housefactory::housefactory()
{
}

```

```

housefactory* housefactory::createHouse(house* h)
{
    housefactory* house;
    switch (h->getType())
    {
        case 0:
            house = new studio(h->getAdress(), h->getCost(), h->getDistrict(), h->getState(), h->getClient());
            break;
        case 1:
            house = new oneroom(h->getAdress(), h->getCost(), h->getDistrict(), h->getState(), h->getClient());
            break;
        case 2:
            house = new tworooms(h->getAdress(), h->getCost(), h->getDistrict(), h->getState(), h->getClient());
            break;
        case 3:
            house = new threerooms(h->getAdress(), h->getCost(), h->getDistrict(), h->getState(), h->getClient());
            break;
        case 4:
            house = new manyrooms(h->getAdress(), h->getCost(), h->getDistrict(), h->getState(), h->getClient());
            break;
    }
    return house;
}

```

Данный класс является «фабрикой» объектов классов квартир
необходимого типа

```

manyrooms.h:
#ifndef MANYROOMS_H
#define MANYROOMS_H

#include <QObject>
#include <housefactory.h>

class manyrooms : public housefactory
{
    Q_OBJECT
public:
    manyrooms();
    manyrooms(QString address, int cost, QString district, int state, client*
person);
};

#endif // MANYROOMS_H

```

```

manyrooms.cpp:
#include "manyrooms.h"

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

manyrooms::manyrooms(QString address, int cost, QString district, int state,
client* person)
{
    this->setType(4);
}

```

```

        this->setAddress(address);
        this->setCost(cost);
        this->setDistrict(district);
        this->setState(state);
        this->setClient(person);
    }

```

Данный класс используется для создания объекта квартиры с 4 и более комнатами

```

oneroom.h:
#ifndef ONEROOM_H
#define ONEROOM_H

#include <QObject>
#include <housefactory.h>

class oneroom : public housefactory
{
    Q_OBJECT
public:
    oneroom();
    oneroom(QString address, int cost, QString district, int state, client*
person);
};

#endif // ONEROOM_H

```

```

oneroom.cpp:
#include "oneroom.h"

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

oneroom::oneroom(QString address, int cost, QString district, int state, client*
person)
{
    this->setType(1);
    this->setAddress(address);
    this->setCost(cost);
    this->setDistrict(district);
    this->setState(state);
    this->setClient(person);
}

```

Данный класс используется для создания объекта квартиры с 1 комнатой

```

studio.h:
#ifndef STUDIO_H
#define STUDIO_H

#include <QObject>
#include <housefactory.h>

class studio : public housefactory
{
    Q_OBJECT
public:
    studio();
}

```



```

        studio(QString address, int cost, QString district, int state, client*
person);
};

#endif // STUDIO_H

```

studio.cpp:

```

#include "studio.h"

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

studio::studio(QString address, int cost, QString district, int state, client*
person)
{
    this->setType(0);
    this->setAddress(address);
    this->setCost(cost);
    this->setDistrict(district);
    this->setState(state);
    this->setClient(person);
}

```

Данный класс используется для создания объекта квартиры-студии

threerooms.h

```

#ifndef THREEROOMS_H
#define THREEROOMS_H

#include <QObject>
#include <housefactory.h>

class threerooms : public housefactory
{
    Q_OBJECT
public:
    threerooms();
    threerooms(QString address, int cost, QString district, int state, client*
person);
};

#endif // THREEROOMS_H

```

threerooms.cpp

```

#include "threerooms.h"

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

threerooms::threerooms(QString address, int cost, QString district, int state,
client* person)
{
    this->setType(3);
    this->setAddress(address);
    this->setCost(cost);
    this->setDistrict(district);
    this->setState(state);
    this->setClient(person);
}

```

```
}
```

Данный класс используется для создания объекта квартиры с 3 комнатами

tworooms.h

```
#ifndef TWOROOMS_H
#define TWOROOMS_H

#include <QObject>
#include <housefactory.h>

class tworooms : public housefactory
{
    Q_OBJECT
public:
    tworooms();
    tworooms(QString address, int cost, QString district, int state, client*
person);
};

#endif // TWOROOMS_H
```

tworooms.cpp

```
#include "tworooms.h"

tworooms::tworooms()
{
    this->setType(2);
}

tworooms::tworooms(QString address, int cost, QString district, int state,
client* person)
{
    this->setType(2);
    this->setAddress(address);
    this->setCost(cost);
    this->setDistrict(district);
    this->setState(state);
    this->setClient(person);
}
```

Данный класс используется для создания объекта квартиры с 2 комнатами

Вывод:

Я смог спроектировать и реализовать простую информационную систему