

КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

«Описание классов и порождение объектов»

по курсу: Объектно-ориентированное программирование

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

подпись, дата

инициалы, фамилия

1. Цель работы

Научиться на практике применять паттерны проектирования.

1.1 Задание

- работа выполняется в графическом приложении Qt
 - используются паттерны (не менее 2-х)
 - для ввода информации пользователем предусматриваются формы для ввода
 - вывод информации (для выбора) осуществляется в виде таблицы (виджет QTableView) - реализована многозначная зависимость
1. Студенту предлагается выбрать предметную область (либо из предложенных вариантов, либо предложить свою) и согласовать ее с преподавателем (преподаватель фиксирует у себя тему) с тем, чтобы в дальнейшем использовать в курсовом проекте по объектно-ориентированному программированию. Не разрешается выбирать одну и ту же предметную область внутри одной группы.
 2. Проанализировать предметную область, выделить сущности предметной области и сформулировать разрабатываемый функционал.
В предметной области должна быть реализована многозначная зависимость.
 3. На основе сущностей предметной области спроектировать иерархии (или совокупности) связанных классов. Для каждого класса в этом разделе должны быть указаны поля и методы (т.е. хедеры классов).
 4. В работе рекомендуется использовать не менее двух паттернов. Паттерны должны быть разных типов: порождающие (кроме Синглтона), поведенческие, структурные. Студент должен обосновать использование тех или иных паттернов. Результатом проектирования является представленная диаграмма классов (или несколько диаграмм).
 5. Собственно, разработка программы. На основе перечисленного функционала (из пункта 2) формируете структуру меню и уделяете внимание каждому пункту.

Вариант 15 – автосалон

Сущность – автомобили

Список доступных Автомобилей							
	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12030	Moonwalk ...	2006	100300	Выдана
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии
3	Carrera GT	Люкс	12032	Moonwalk ...	2006	180540	В наличии
4	Carrera GT	Электро	12033	Moonwalk ...	2006	160480	В наличии

Сущность – покупатели

Информация о покупателе				
	Паспорт	ФИО	EMail	Номер
1	1234567891	Калинин С.К.	bimbibbam...	12345
2	1234567892	Калинин С.К.	bimbibbam...	12345
3	1234567893	Калинин С.К.	bimbibbam...	12345

Выдача автомобилей

История операций				
	Паспорт	Серийный номер	Дата выдачи	Дата возврата
1	1234567891	12030	22.05.2023, ...	
2	1234567893	12032	22.05.2023, ...	22.05.2023, ...

Логика приложения:

Автосалон.

Автомобили:

Модель – вводится любая модель автомобиля

Серийный номер – идентификатор автомобиля, должен содержать только цифры, у каждой машины должен быть свой индивидуальный номер

Комплектация – выбирается 1 из 4 вариантов и влияет на величину первого взноса

Цвет – вводится любой цвет

Год производства – влияет на величину первого взноса

Добавить – добавление автомобиля в список доступных авто

Последний запрос – ввод в поля авто данных предыдущего автомобиля

Покупатели:

Паспортные данные – у каждого покупателя данные должны быть индивидуальными, вводятся только цифры

ФИО - вводятся данные покупателя

Электронная почта - вводятся данные покупателя

Телефон - вводятся данные покупателя, только цифры

Добавить клиента – добавление клиента в список покупателей

Выдача автомобилей:

Дата выдачи – вводится дата выдачи


Дата возврата – вводится дата возврата

Выдать – в область истории операций вводится выбранный автомобиль, покупатель и введенная дата выдачи

Вернуть - в область истории операций вводится дата возврата выбранной операции

2. Форма

Захаров ЛР8



PORSCHE

Модель: Carrera GT

Серийный номер: 12033

Комплектация: Электро

Цвет: Moonwalk Grey

Год производства: 2006

первый взнос: 160480

Паспортные данные: 1234567893

ФИО: Калинин С.К.

электронная почта: mbam@gmail.com

Телефон: 12345

Добавить клиента

Добавить

Последний запрос

Дата выдачи: 22.05.2023, 10:00

Дата возврата: 22.05.2023, 15:00

Выдать

Вернуть

Список доступных Автомобилей


	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12030	Moonwalk ...	2006	100300	Выдана
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии
3	Carrera GT	Люкс	12032	Moonwalk ...	2006	180540	В наличии
4	Carrera GT	Электро	12033	Moonwalk ...	2006	160480	В наличии

Информация о покупателе

Паспорт	ФИО	E-Mail	Номер
1	Калинин С.К.	bimbibam...	12345
2	Калинин С.К.	bimbibam...	12345
3	Калинин С.К.	bimbibam...	12345

История операций

Паспорт	Серийный номер	Дата выдачи	Дата возврата
1	1234567891	12030	22.05.2023, ...
2	1234567893	12032	22.05.2023, ...



Label_10: passport

Label_11: FIOclient

Label_12: EMail

Label_13: phonenumber

btnAddClient

Label_17

Label: Brand

Label_2: Number

Label_3: CarType

Label_4: Colour

Label_5: Age

Label_7: cost

btnCacl btnUndo

Label_16

Label_14: issuedate

Label_15: refunddate

btnissue btnrefund

clienttable

refundtable

tableView

Были использованы следующие виджеты:

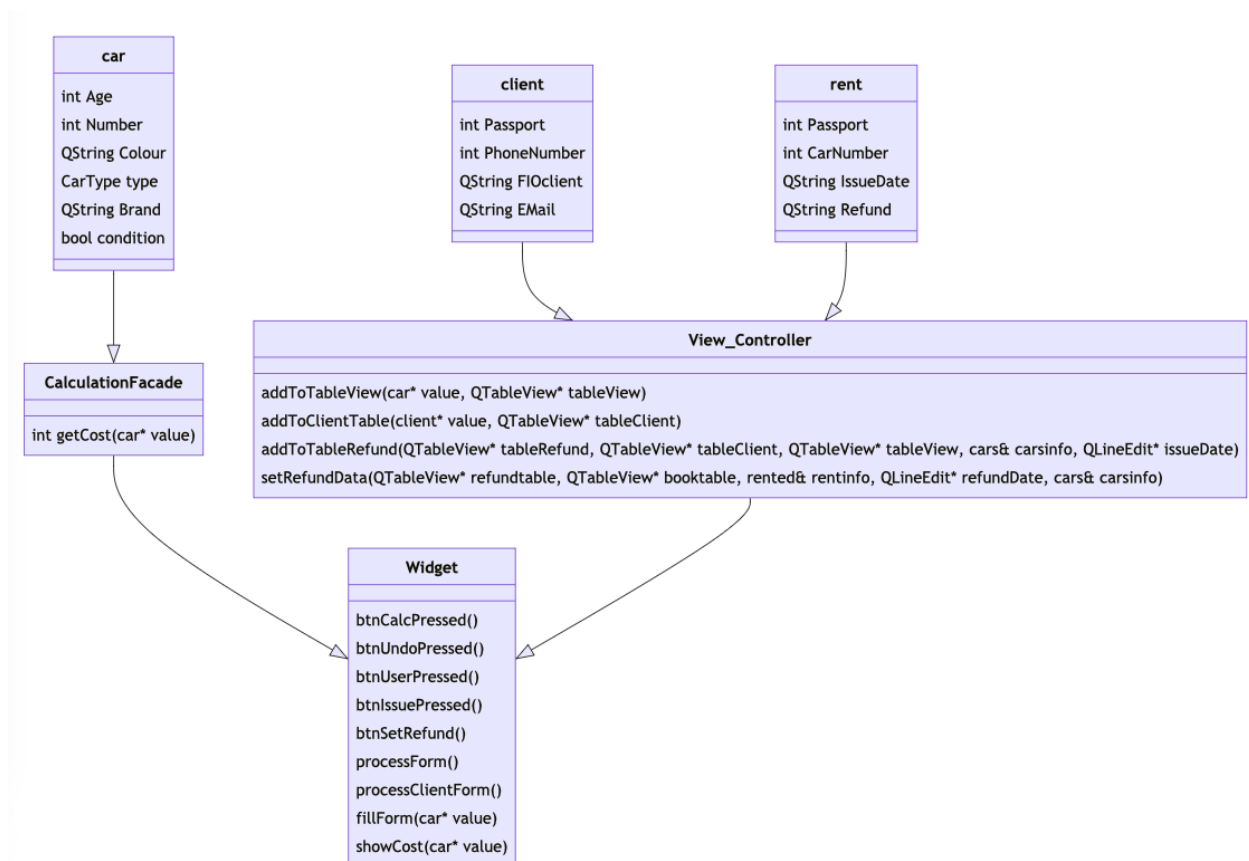
19 – label: cost, label, Label_2, Label_3, Label_4, Label_5, Label_6, Label_7, Label_8, Label_9, Label_10, Label_11, Label_12, Label_13, Label_14, Label_15, Label_16, Label_17 которые относятся к наименованию, пояснительным надписям для полей, выводу первого взноса

5 – pushButton: btnCalc, btnUndo, btnAddClient, btnissue, btnrefund, которые отвечают за ввод нового авто в список, ввод в форму данных предыдущего автомобиля, добавление клиента в список, фиксирование выдачи и возврата автомобилей

1 – comboBox: CarType для выбора класса автомобиля

10 – lineEdit: age, brand, number, Email, fioclient, passport, phonenumber, issuedate, refunddate: ввод индивидуальных значений автомобилей и клиентов

3. Диаграммы:



Класс car содержит информацию о машине, включая возраст, номер, цвет, тип и бренд.

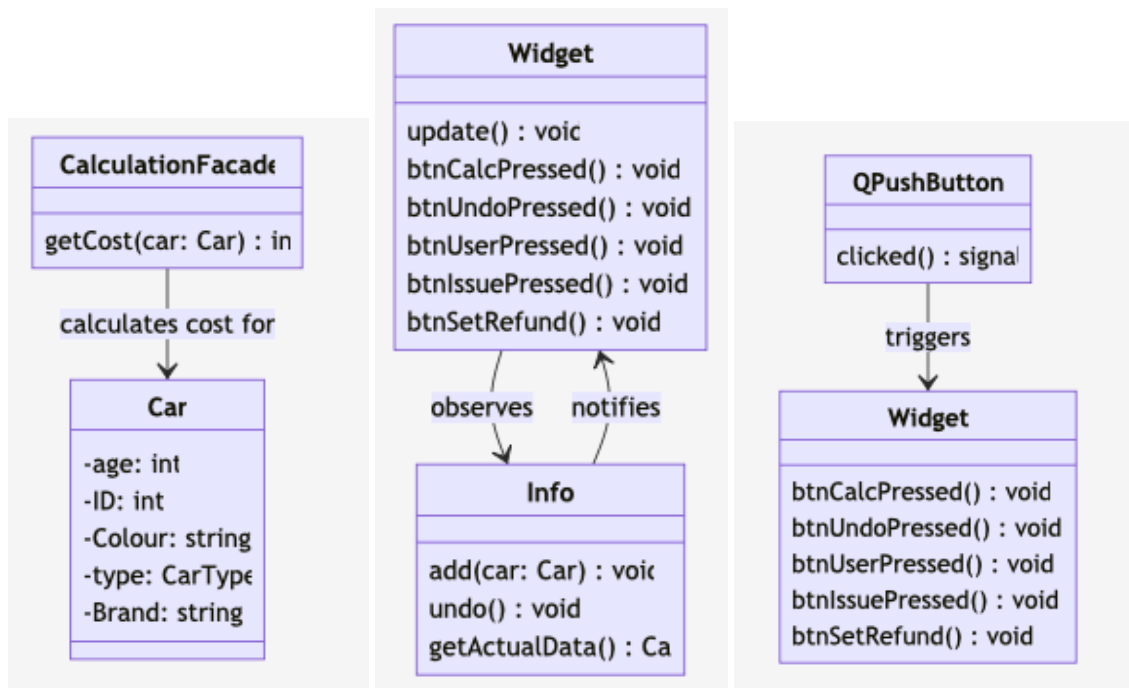
Класс client содержит информацию о клиенте, включая паспорт, номер телефона, ФИО и электронную почту.

Класс rent содержит информацию об аренде, включая паспорт, номер машины, дату выдачи и возврата.

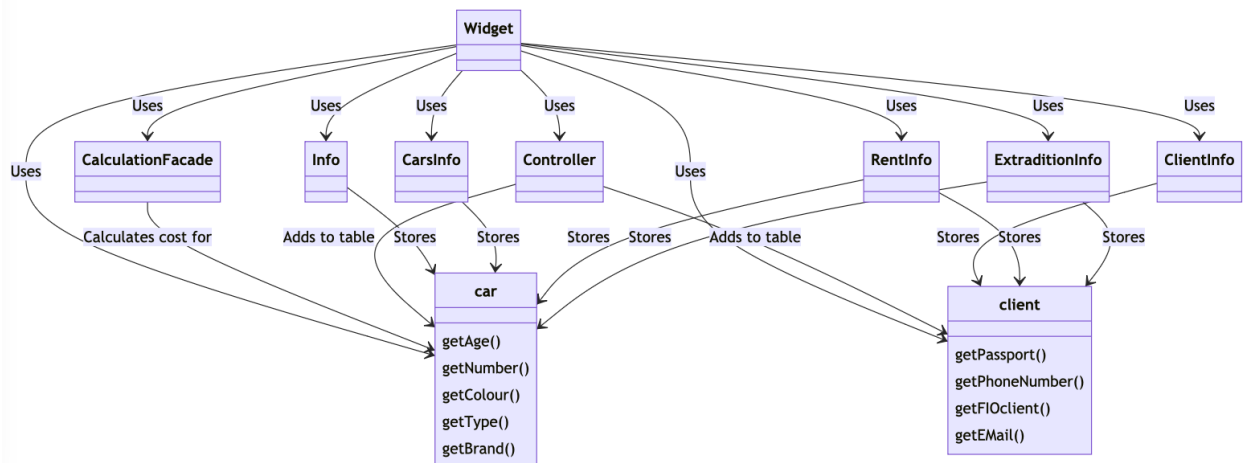
Класс CalculationFacade предоставляет метод `getCost`, который вычисляет стоимость аренды автомобиля.

Класс View_Controller предоставляет методы для добавления данных в различные представления (таблицы).

Класс Widget предоставляет слоты для обработки нажатий кнопок и методы для обработки данных формы.



общая диаграмма классов, описывающая проект приложения:



класс **Widget** использует классы **CalculationFacade**, **car**, **client**, **Controller**, **Info**, **CarsInfo**, **ClientInfo**, **RentInfo** и **ExtraditionInfo**. Класс **CalculationFacade** используется для вычисления стоимости для класса **car**. Класс **Controller** добавляет объекты классов **car** и **client** в таблицу. Классы **Info**, **CarsInfo**, **ClientInfo**, **RentInfo** и **ExtraditionInfo** хранят объекты классов **car** и **client**. Классы **car** и **client** имеют методы для получения своих свойств.

4.1 Текст программы (abstcalc.h)

```
#ifndef ABSTRACTCALC_H
```

```
#define ABSTRACTCALC_H
```

```
#include "car.h"
```

```
//абстрактный класс для создания объектов классов расчета
```

```

class AbstractCalc
{
public:
    AbstractCalc();
    virtual int getCost(car *value)=0;
    virtual ~AbstractCalc(){};
};

// Класс создаваемый для расчета стоимости стандартной комплектации
// все классы имеют родителя AbstractCalc и являются объектами конечных типов
class StandardCalc : public AbstractCalc
{
public:
    int getCost(car *value){
        return (((value->getAge()))*50);
    }
};

// Класс создаваемый для расчета стоимости для авто комфорт класса
class ComfortCalc : public AbstractCalc
{
public:
    int getCost(car *value){
        return (((value->getAge()))*70);
    }
};

// Класс создаваемый для расчета стоимости для люкс
class LuxuryCalc : public AbstractCalc
{
public:
    int getCost(car *value){
        return (((value->getAge()))*90);
    }
};

// Класс создаваемый для расчета стоимости для электро
class ElectricCalc : public AbstractCalc

```

```

{
public:
    int getCost(car *value){
        return (((value->getAge()))*80);
    }
};

#endif // ABSTRACTCALC_H

```

Текст программы (calcfactory.h)

```

#ifndef CALCFACTORY_H
#define CALCFACTORY_H

#include "car.h"
#include "abstractcalc.h"

class CalcFactory
{
public:
    virtual AbstractCalc * fabrica()=0;
    virtual ~CalcFactory(){};
};

// для каждой комплектации переопределяем функцию fabrica(), которая создает объект класса
// вычисления стоимости

// все классы имеют родителя CalcFactory и называются фабричными

class StandardFactory : public CalcFactory
{
public:
    AbstractCalc * fabrica(){return new StandardCalc;}
};

class ElectricFactory: public CalcFactory
{
public:
    AbstractCalc * fabrica(){return new ElectricCalc;}
};

class ComfortFactory: public CalcFactory
{

```


public:

```
    AbstractCalc * fabrica(){return new ComfortCalc;}
```

```
};
```

```
class LuxuryFactory: public CalcFactory
```

```
{
```

public:

```
    AbstractCalc * fabrica(){return new LuxuryCalc;}
```

```
};
```

```
#endif // CALCFACTORY_H
```

Текст программы (calculationfacade.h)

```
#ifndef CALCULATIONFACADE_H
```

```
#define CALCULATIONFACADE_H
```

```
#include <QObject>
```

```
#include <car.h>
```

```
#include "calcfactory.h"
```

```
class CalculationFacade : public QObject//базовым классом является класс QOBJECT
```

```
{
```

```
    Q_OBJECT
```

public:

```
    explicit CalculationFacade(QObject *parent = nullptr);//конструктор
```

```
    static int getCost(car *value);//функция получения стоимости
```

signals:

```
};
```

```
#endif // CALCULATIONFACADE_H
```

Текст программы (car.h)

```
#ifndef CAR_H
```

```
#define CAR_H
```

```
#include <QObject>
```

```
class car : public QObject //базовым классом является класс QOBJECT
```

```
{
```

```
    Q_OBJECT
```

public:

```
    enum CarType { //комплектации
```

```

        STANDARD,
        COMFORT,
        LUXURY,
        ELECTRIC
    };

    explicit car(int inputAge, int inputNumber,QString Colour, CarType inputCarType,QString
inputBrand,QObject *parent = nullptr);//конструктор

    int getAge();
    int getNumber();
    QString getColour();
    CarType getType();//функции получения private данных из класса
    QString getBrand();
    QString getTypeString();
    bool condition;
    int Number;
private:
    int Age;//поля для записи данных с формы
    QString Colour;
    CarType Type;
    QString Brand;
};

#endif // CAR_H

Текст программы (cars.h)

#ifndef CARS_H
#define CARS_H

#include <QObject>
#include <car.h>

class cars : public QObject//базовым классом является класс QOBJECT
{
    Q_OBJECT
public:
    explicit cars(QObject *parent = nullptr);
    ~cars();

    void undo();//функция манипулирует над actualdata добавляя в нее значение или null

```

```

bool hasCars();//наличие элементов в коллекции

car *getActualData();//функция возвращающая последний элемент коллекции

void add(car *value);//добавление элемента в коллекцию

QList<car *> array;//список в котором храняться элементы

private:

    car *actualData;//последний элемент коллекции

signals:

    void notifyObservers();//сигнал наблюдателю

};

#endif // CARS_H

```

Текст программы (carsuse.h)

```

#ifndef CARSUSE_H
#define CARSUSE_H

#include <QObject>
#include <caruse.h>

class carsuse : public QObject//базовым классом является класс QOBJECT
{
    Q_OBJECT

public:
    explicit carsuse(QObject *parent = nullptr);
    ~carsuse();

    void undo();//функция манипулирует над actualdata добавляя в нее значение или null

    bool hasCarsuse();//наличие элементов в коллекции

    caruse *getActualData();//функция возвращающая последний элемент коллекции

    void add(caruse *value);//добавление элемента в коллекцию

private:

    QList<caruse *> array;//список в котором храняться элементы

    caruse *actualData;//последний элемент коллекции

signals:

    void notifyObservers();//сигнал наблюдателю

};

#endif // CARSUSE_H

```

Текст программы (caruse.h)

```

#ifndef CARUSE_H

```

```

#define CARUSE_H

#include <QObject>

class caruse : public QObject //базовым классом является класс QObject
{
    Q_OBJECT
public:
    enum CarType { //комплектации
        STANDARD,
        COMFORT,
        LUXURY,
        ELECTRIC
    };

    explicit caruse(int inputAge, int inputNumber,QString Colour, CarType inputCarType,QString
inputBrand,QObject *parent = nullptr); //конструктор

    int getAge();
    int getNumber();
    QString getColour();
    CarType getType(); //функции получения private данных из класса
    QString getBrand();
    QString getTypeString();
private:
    int Age; //поля для записи данных с формы
    int Number;
    QString Colour;
    CarType Type;
    QString Brand;
};

#endif // CARUSE_H

```

Текст программы (client.h)

```

#ifndef CLIENT_H
#define CLIENT_H
#include <QObject>
class client : public QObject
{

```

```

Q_OBJECT

public:
    explicit client(QObject *parent = nullptr);

    explicit client(int inputPassport, int inputPhoneNumber,QString inputFIO,QString inputEMail,QObject
*parent = nullptr);//конструктор

    int getPassport();

    int getPhoneNumber();

    QString getFIO();//функции получения private данных из класса

    QString getEMail();

private:
    int Passport;//поля для записи данных с формы

    int PhoneNumber;

    QString FIO;

    QString EMail;

signals:
};

class clients : public QObject//базовым классом является класс QOBJECT
{
    Q_OBJECT

public:
    explicit clients(QObject *parent = nullptr);

    ~clients();

    //void undo();//функция манипулирует над actualdata добавляя в нее значение или null

    bool hasClients();//наличие элементов в коллекции

    //car *getActualData();//функция возвращающая последний элемент коллекции

    void add(client *value);//добавление элемента в коллекцию

    QList<client *> array;//список в котором храняться элементы класса estate

    //car *actualData;//последний элемент коллекции

signals:
    //void notifyObservers();//сигнал наблюдателю

};

#endif // CLIENT_H

Текст программы (clients.h)

#ifndef CLIENTS_H

```

```

#define CLIENTS_H

#include <QObject>

class clientss : public QObject
{
    Q_OBJECT

public:
    explicit clientss(QObject *parent = nullptr);

signals:
};

#endif // CLIENTS_H

```

Текст программы (rent.h)

```

#ifndef RENT_H
#define RENT_H

#include <QObject>

class rent : public QObject
{
    Q_OBJECT

public:
    explicit rent(int inputPassport, int inputCarNumber, QString Issue, QObject *parent =
    nullptr); //конструктор

    int getPassport();

    int getCarNumber();

    QString getIssue(); //функции получения private данных из класса

    QString getRefund();

    void setRefund(QString inputRefund);

    int CarNumber;

private:
    int Passport; //поля для записи данных с формы

    QString Issue;

    QString Refund;

};

class rented : public QObject //базовым классом является класс QOBJECT
{
    Q_OBJECT

```

```

public:
    explicit rented(QObject *parent = nullptr);
    ~rented();

    bool hasrented();//наличие элементов в коллекции

    void add(rent *value);//добавление элемента в коллекцию

    QList<rent *> array;//список в котором храняться элементы класса estate
};

#endif // RENT_H

```

Текст программы (viewcontrol.h)

```

#ifndef VIEWCONTROL_H
#define VIEWCONTROL_H

#include <QObject>
#include <rent.h>
#include <car.h>
#include <cars.h>
#include <client.h>
#include <QTableView>
#include <QStandardItemModel>
#include <calculationfacade.h>
#include <QLineEdit>

class View_Controller : public QObject
{
    Q_OBJECT

public:
    explicit View_Controller(QObject *parent = nullptr);

    void addToTableView(car* lastObject, QTableView* tableView);

    void addToClientTable(client* Object, QTableView* tableClient);

    rent* addToTableRefund(QTableView* tableRefund, QTableView* tableClient, QTableView*
tableView, cars& carsinfo, QLineEdit* issueDate);

    void setRefundData(QTableView* refundtable, QTableView* cartable, rented& rentinfo, QLineEdit*
refundDate, cars& carsinfo);//лайн эдит с датой, список выданных, тблица авто

signals:
};

#endif // VIEWCONTROL_H

```

Текст программы (widget.h)

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <cars.h>
#include <car.h>
#include <client.h>
#include <calculationfacade.h>
#include <QStandardItemModel>
#include <qtableview.h>
#include <rent.h>
#include <view_controller.h>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    clients clientinfo;
    cars carsinfo;
    rented rentinfo;

public slots:
    void update();//функция вызываемая при передаче сигнала наблюдателю
    //вызывается при поступлении сигнала, после выполняется
    //взаимодействие с States и заполнение данных на форме.

private slots:
    void btnCalcPressed();//слот выполняющийся при нажатии кнопки "расчитать стоимость"
    void btnUndoPressed();//слот выполняющийся при нажатии кнопки "последний запрос"
    void btnUserPressed();
    void btnIssuePressed();
    void btnSetRefund();

private:

```



```

Ui::Widget *ui;

car *processForm();//функция обрабатывающая данные формы, создает объект класс
client *processClientForm();

void fillForm(car *value);//функция отвечающая за отображение данных объекта класса на форме

QString showCost(car *value);//функция отображающая стоимость получая ее от класса
calculationfacade

cars info;//коллекция предыдущих запросов

void addToTableView(car* lastObject, QTableView* tableView);

void addToUserTable(client* Object, QTableView* tableUser);

rent* addToTableRefund(QTableView* tableRefund);

void setRefundData(QTableView* refundtable);

View_Controller controller;

//void moveToUsedTable();

//void moveToTableView();

};

#endif // WIDGET_H

```

Текст программы (calculationfacade.cpp)

```

#include "calculationfacade.h"

CalculationFacade::CalculationFacade(QObject *parent)
    : QObject{parent}
{
}

int CalculationFacade::getCost(car *value) {
// Путь создания объектов:CalcFactory->***Factory->***Calc; ***Calc вызывает функцию getCost()
int cost;//переменная с ценой
switch (value->getType())//определяем тип комплектации
{
    case car::CarType::STANDARD:{
        //создается класс фабрика

        CalcFactory * standard_factory = new StandardFactory; // выделяем память под объект класса
        StandardFactory

        //создание объекта фабричного класса и вычисление стоимсоти

        cost = standard_factory->fabrica()->getCost(value); // создаем объект и рассчитываем его
        стоимость

        delete standard_factory; // очищаем память
    }
}
}

```

```

        break;
    }
    case car::CarType::COMFORT:{
        CalcFactory * comfort_factory = new ComfortFactory; // выделяем память под объект класса
        ComfortFactory

        cost = comfort_factory->fabrica()->getCost(value); // создаем объект и рассчитываем его
        стоимость

        delete comfort_factory; // очищаем память

        break;
    }
    case car::CarType::LUXURY:{
        CalcFactory * luxury_factory = new LuxuryFactory; // выделяем память под объект класса
        LuxuryFactory

        cost = luxury_factory->fabrica()->getCost(value); // создаем объект и рассчитываем его стоимость

        delete luxury_factory; // очищаем память

        break;
    }
    case car::CarType::ELECTRIC:{
        CalcFactory * electric_factory = new ElectricFactory; // выделяем память под объект класса
        ElectricFactory

        cost = electric_factory->fabrica()->getCost(value); // создаем объект и рассчитываем его
        стоимость

        delete electric_factory; // очищаем память

        break;
    }
    default:
        cost = -1;

        break;
    }

    return cost;
}

```

Текст программы (car.cpp)

```

#include "car.h"

car::car(int inputAge, int inputNumber,QString inputColour, CarType inputType,QString
inputBrand,QObject *parent)//конструктор

:QObject{parent}

```

```

{
    Age=inputAge;
    Number=inputNumber;
    Colour=inputColour;
    Type=inputType;
    Brand=inputBrand;
    condition=true;
}

int car::getNumber(){//номер
    return Number;
}

QString car::getColour(){//цвет
    return Colour;
}

car::CarType car::getType(){//комплектация
    return Type;
}

QString car::getTypeString(){
    switch (getType())//определяем тип комплектации
    {
        case car::CarType::STANDARD:{
            return "Стандарт";
            break;
        }
        case car::CarType::COMFORT:{
            return "Комфорт";
            break;
        }
        case car::CarType::LUXURY:{
            return "Люкс";
            break;
        }
        case car::CarType::ELECTRIC:{
            return "Электро";

```

```

        break;
    }
}

QString car::getBrand(){//покупатель
    return Brand;
}

int car::getAge(){//возраст
    return Age;
}

```

Текст программы (cars.cpp)

```

#include "cars.h"

cars::cars(QObject *parent)
    : QObject{parent}
{
    actualData = nullptr;
}

cars::~cars()
{
    if(actualData){//удаление actualdata
        delete actualData;
        actualData=nullptr;
    }

    array.clear();//удаление о очистка array
    qDeleteAll(array);
}

bool cars::hasCars(){
    return !array.empty();//в коллекции есть элементы
}

car *cars::getActualData(){
    return array.back();
}

void cars::add(car *value){
    array.append(value);//добавление элемента в коллекцию
}

```

```

}

void cars::undo(){
    if(!hasCars()||(array.size()==1)){
        actualData=nullptr;//если в коллекции нет элементов actualdata равна nullptr
    }
    else {//если есть элементы то заполняем actualdata значением и отправляем сигнал наблюдателю
        actualData=getActualData();
        array.removeLast();
        emit notifyObservers();//сигнал уведомляющий наблюдателя
    }
}

```

Текст программы (carsuse.cpp)

```

#include "carsuse.h"

carsuse::carsuse(QObject *parent)
    : QObject{parent}
{
    actualData = nullptr;
}

carsuse::~carsuse()
{
    if(actualData){//удаление actualdata
        delete actualData;
        actualData=nullptr;
    }
    array.clear();//удаление о очистка array
    qDeleteAll(array);
}

bool carsuse::hasCarsuse(){
    return !array.empty();//в коллекции есть элементы
}

caruse *carsuse::getActualData(){
    return array.back();
}

void carsuse::add(caruse *value){
    array.append(value);//добавление элемента в коллекцию
}

```

```

}

void caruse::undo(){
    if(!hasCarsuse()||(array.size()==1)){
        actualData=nullptr;//если в коллекции нет элементов actualdata равна nullptr
    }
    else {//если есть элементы то заполняем actualdata значением и отправляем сигнал наблюдателю
        actualData=getActualData();
        array.removeLast();
        emit notifyObservers();//сигнал уведомляющий наблюдателя
    }
}

```

Текст программы (caruse.cpp)

```

#include "caruse.h"

caruse::caruse(int inputAge, int inputNumber,QString inputColour, CarType inputType,QString
inputBrand,QObject *parent)//конструктор
    :QObject{parent}
{
    Age=inputAge;
    Number=inputNumber;
    Colour=inputColour;
    Type=inputType;
    Brand=inputBrand;
}

int caruse::getNumber(){//номер
    return Number;
}

QString caruse::getColour(){//цвет
    return Colour;
}

caruse::CarType caruse::getType(){//комплектация
    return Type;
}

QString caruse::getTypeString(){
    switch (getType())//определяем тип комплектации
    {

```

```

        case caruse::CarType::STANDARD:{
            return "Стандарт";
            break;
        }
        case caruse::CarType::COMFORT:{
            return "Комфорт";
            break;
        }
        case caruse::CarType::LUXURY:{
            return "Люкс";
            break;
        }
        case caruse::CarType::ELECTRIC:{
            return "Электро";
            break;
        }
    }
}

```

```

QString caruse::getBrand()//покупатель
    return Brand;

```

```

}

int caruse::getAge()//возраст
    return Age;
}

```

Текст программы (client.cpp)

```

#include "client.h"

```

```

client::client(QObject *parent)
    : QObject{parent}

```

```

{
}

```

```

client::client(int inputPassport, int inputPhoneNumber,QString inputFIO, QString inputEMail,QObject
*parent)//конструктор

```

```

    :QObject{parent}
{

```

```

    Passport=inputPassport;
    PhoneNumber=inputPhoneNumber;
    FIO=inputFIO;
    EMail=inputEMail;
}
int client::getPassport(){//паспорт
    return Passport;
}
QString client::getFIO(){//фio
    return FIO;
}
QString client::getEMail(){//покупатель почта
    return EMail;
}
int client::getPhoneNumber(){//телефон
    return PhoneNumber;
}
clients::clients(QObject *parent)
    : QObject{parent}
{
}
clients::~clients()
{
    array.clear();//удаление о очистка array
    qDeleteAll(array);
}
bool clients::hasClients(){
    return !array.empty();//в коллекции есть элементы
}
void clients::add(client *value){
    array.append(value);//добавление элемента в коллекцию
}

```

Текст программы (rent.cpp)

```
#include "rent.h"
```



```

rent::rent(int inputPassport, int inputCarNumber,QString inputIssue,QObject *parent)//конструктор
    :QObject{parent}
{
    Passport=inputPassport;
    CarNumber=inputCarNumber;
    Issue=inputIssue;
    Refund = " ";
}

int rent::getPassport(){//паспорт
    return Passport;
}

QString rent::getIssue(){//выдача
    return Issue;
}

QString rent::getRefund(){//возвр
    return Refund;
}

int rent::getCarNumber(){//возраст
    return CarNumber;
}

void rent::setRefund(QString inputRefund){
    Refund=inputRefund;
}

rented::rented(QObject *parent)
    : QObject{parent}
{
}

rented::~~rented()
{
    array.clear();//удаление о очистка array
    qDeleteAll(array);
}

bool rented::hasrented(){
    return !array.empty();//в коллекции есть элементы
}

```

```

}

void rented::add(rent *value){
    array.append(value);//добавление элемента в коллекцию
}

```

Текст программы (viewcontrol.cpp)

```

#include "viewcontrolr.h"

ViewControl::ViewControl(QObject *parent)
    : QObject{parent}
{
}

void ViewControl::addToTableView(car* lastObject, QTableView* tableView)//cool
{
    CalculationFacade cur;

    // Получаем указатель на модель данных
    QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableView->model());
    if (!model)
    {
        // Если модель данных не является типом QStandardItemModel, создаем новую модель
        model = new QStandardItemModel(tableView);
        tableView->setModel(model);

        model->setHorizontalHeaderLabels({ "Название", "Комплектация", "Номер", "Цвет", "Год
выпуска", "Взнос", "Наличие" });
    }

    // Получаем количество строк в таблице
    int rowCount = model->rowCount();

    // Создаем новую строку в модели данных
    QList<QStandardItem*> newRow;

    // Создаем элементы для каждого столбца таблицы
    QString condition = "В наличии";

    QStandardItem* typeItem = new QStandardItem(lastObject->getTypeString());
    QStandardItem* ageItem = new QStandardItem(QString::number(lastObject->getAge()));
    //QStandardItem* residentsItem = new QStandardItem(QString::number(lastObject->getResidents()));
    //QStandardItem* monthsItem = new QStandardItem(QString::number(lastObject-
>getMonthsForMVC()));

    //QStandardItem* priceItem = new QStandardItem(QString::number(lastObject->getPrice()));
}

```

```

QStandardItem* BrandItem = new QStandardItem(lastObject->getBrand());
QStandardItem* ColourItem = new QStandardItem(lastObject->getColour());
QStandardItem* NumberItem = new QStandardItem(QString::number(lastObject->getNumber()));
//QStandardItem* TitleItem = new QStandardItem(lastObject->getTitle());
QStandardItem* CostItem = new QStandardItem(QString::number(cur.getCost(lastObject)));
QStandardItem* conditionItem = new QStandardItem(condition);

// Добавляем созданные элементы в новую строку
newRow.append(BrandItem);
newRow.append(typeItem);
//newRow.append(priceItem);
newRow.append(NumberItem);
newRow.append(ColourItem);
newRow.append(ageItem);
newRow.append(CostItem);
newRow.append(conditionItem);

// Добавляем новую строку в модель данных
model->insertRow(rowCount, newRow);

// Обновляем таблицу
tableView->viewport()->update();
}

void View_Controller::addToClientTable(client* Object,QTableView* tableClient){//cool
    // Получаем указатель на модель данных
    QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableClient->model());
    if (!model)
    {
        // Если модель данных не является типом QStandardItemModel, создаем новую модель
        model = new QStandardItemModel(tableClient);
        tableClient->setModel(model);
        model->setHorizontalHeaderLabels({"Паспорт","ФИО","EMail","Номер"});
    }
    int rowCount = model->rowCount();

    // Создаем новую строку в модели данных
    QList<QStandardItem*> newRow;

    // Создаем элементы для каждого столбца таблицы

```

```

QStandardItem* PassportItem = new QStandardItem(QString::number(Object->getPassport()));
QStandardItem* FIOItem = new QStandardItem(Object->getFIO());
QStandardItem* EMailItem = new QStandardItem(Object->getEMail());
QStandardItem* PhoneNumberItem = new QStandardItem(QString::number(Object-
>getPhoneNumber()));

// Добавляем созданные элементы в новую строку
newRow.append(PassportItem);
newRow.append(FIOItem);
//newRow.append(priceItem);
newRow.append(EMailItem);
newRow.append(PhoneNumberItem);

// Добавляем новую строку в модель данных
model->insertRow(rowCount, newRow);

// Обновляем таблицу
tableClient->viewport()->update();
}

rent* View_Controller::addToTableRefund(QTableView* tableRefund,QTableView*
tableClient,QTableView* tableView,cars& carsinfo,QLineEdit* issueDate){//остальные две таблицы
тоже нужны, и список книг и лайн эдит с датой

    int curid;

    QString curids;

// Получаем выделенные элементы из двух предыдущих таблиц
    QModelIndexList selectedPersonIndexes = tableClient->selectionModel()->selectedIndexes();
    QModelIndexList selectedCarIndexes = tableView->selectionModel()->selectedIndexes();
    QModelIndex index = tableView->currentIndex();

//получаем указатель на модель данных для замены поля наличия
    QStandardItemModel* model1 = dynamic_cast<QStandardItemModel*>(tableView->model());

// Изменяем значение в четвертом столбце выбранной строки
    QStandardItem* item = model1->itemFromIndex(index.sibling(index.row(), 6)); // 3 - индекс
четвертого столбца

// Получаем номер читательского билета и идентификатор книги из выделенных элементов
    QString personNumber = selectedPersonIndexes.at(0).data().toString();
    QString carNumber = selectedCarIndexes.at(2).data().toString();

    for(int i =0;i<carsinfo.array.size();i++){
        curid=carsinfo.array[i]->Number;
    }
}

```

```

    curids=QString::number(curid);
    if(curids==carNumber){
        if (carsinfo.array[i]->condition==false){    //проверяете состояние автомобиля, используя
condition
                                                    //Если автомобиль уже взят в аренду (т.е. condition == true),
                                                    //не разрешаем его повторное взятие, и функция возвращает
управление, и новая запись о выдаче в таблицу не добавляется.

            return nullptr;
        }
        else{
            carsinfo.array[i]->condition=false;
            item->setData("Выдана", Qt::DisplayRole);
            //selectedCarIndexes.at(6).data()="Выдана";
            //ui->clientTable->selectionModel()->selectedIndexes().at(6).data()=cond;
            break;
        }
    }
}

// Получаем данные для поля "Дата выдачи" из LineEdit
QString issueDate1 = issueDate->text();
//extradition* value (personNumber.toInt(),carNumber.toInt(),issueDate);
QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableRefund->model());
if (!model)
{
    // Если модель данных не является типом QStandardItemModel, создаем новую модель
    model = new QStandardItemModel(tableRefund);
    tableRefund->setModel(model);

    model->setHorizontalHeaderLabels({ "Паспорт", "Серийный номер", "Дата выдачи", "Дата
возврата" });
}

int rowCount = model->rowCount();

// Создаем новую строку в модели данных
QList<QStandardItem*> newRow;

// Создаем объекты для хранения данных
QString returnDate = ""; // Поле "Дата возврата" не заполняется

```

```

QStandardItem* personNumberItem = new QStandardItem(personNumber);
QStandardItem* carNumberItem = new QStandardItem(carNumber);
QStandardItem* issueDateItem = new QStandardItem(issueDate1);
QStandardItem* returnDateItem = new QStandardItem(returnDate);
// Добавляем созданные элементы в новую строку
newRow.append(personNumberItem);
newRow.append(carNumberItem);
newRow.append(issueDateItem);
newRow.append(returnDateItem);
// Добавляем новую строку в модель данных
model->insertRow(rowCount, newRow);
tableRefund->viewport()->update();
tableView->viewport()->update();
return new rent(personNumber.toInt(),carNumber.toInt(),issueDate1);
}

void View_Controller::setRefundData(QTableView* refundtable,QTableView* booktable,rented&
rentinfo,QLineEdit* refundDate,cars& carsinfo){//лайн эдит с датой, список выдач, тблиця книг

// Получаем индекс выбранной строки в таблице
QModelIndex index = refundtable->currentIndex();

// Получаем значение из QLineEdit
QString value = refundDate->text();////////////////////////////////////

// Получаем модель данных, которая отображается в таблице
QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(refundtable->model());
QModelIndexList selectedCarIndexes = refundtable->selectionModel()->selectedIndexes();
QString carNumber1 = selectedCarIndexes.at(1).data().toString();
int curid;
for(int i =0;i<rentinfo.array.size();i++){
    curid=rentinfo.array[i]->CarNumber;
    //curids=QString::number(curid);
    if(curid==carNumber1.toInt()){
        if (rentinfo.array[i]->getRefund()==" "){
            rentinfo.array[i]->setRefund(value);
            //item->setData("Выдана", Qt::DisplayRole);
            //selectedBookIndexes.at(6).data()="Выдана";

```

```

        //ui->userTable->selectionModel()->selectedIndexes().at(6).data()=cond;
        break;
    }
}
}

curid=0;

//QModelIndexList selectedBookIndexes = booktable->selectionModel()->selectedIndexes();
//QString bookId = selectedBookIndexes.at(2).data().toString();
for(int i =0;i<carsinfo.array.size();i++){
    curid=carsinfo.array[i]->Number;
    //curids=QString::number(curid);
    if(curid==carNumber1.toInt()){
        if (carsinfo.array[i]->condition==true){           //после возврата автомобиля обновляется
состояние автомобиля

                                                    //Теперь, этот автомобиль можно вновь выдать в аренду.

            return;
        }
    }
    else{
        carsinfo.array[i]->condition=true;
        //item->setData("Выдана", Qt::DisplayRole);
        //selectedBookIndexes.at(6).data()="Выдана";
        //ui->userTable->selectionModel()->selectedIndexes().at(6).data()=cond;
        break;
    }
}

}

// Изменяем значение в четвертом столбце выбранной строки
QStandardItem* item = model->itemFromIndex(index.sibling(index.row(), 3)); // 3 - индекс
четвертого столбца

item->setData(value, Qt::DisplayRole);

// Получаем идентификатор машины из выбранной строки в таблице выдач
QModelIndex indexIssued = refundtable->currentIndex();

QString carNumber2 = indexIssued.sibling(indexIssued.row(), 1).data().toString(); // 1 - индекс
второго столбца

// Находим элемент в таблице машин с таким же идентификатором

```

```

QStandardItemModel* modelCars = dynamic_cast<QStandardItemModel*>(booktable->model());
int rowCount = modelCars->rowCount();
int bookRow = -1;
for (int i = 0; i < rowCount; i++) {
    QModelIndex indexBook = modelCars->index(i, 2); // 0 - индекс первого столбца
    if (indexBook.data().toString() == carNumber2) {
        bookRow = i;
        break;
    }
}

// Если элемент найден, то изменяем значение в шестом столбце
if (bookRow != -1) {
    QStandardItem* item = modelCars->itemFromIndex(modelCars->index(bookRow, 6)); // 5 - индекс
    шестого столбца (наличие)
    item->setData("В наличии", Qt::DisplayRole);

    // Обновляем модель данных в таблице машин
    booktable->setModel(modelCars);
}

// Обновляем модель данных в таблице
refundtable->setModel(model);
}

```

Текст программы (widget.cpp)

```

#include "widget.h"
#include "ui_widget.h"
#include <QPixmap>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
    , info(this)
{
    ui->setupUi(this);
    ui->btnUndo->setEnabled(false);

    QPixmap pix("/Users/andrey/Documents/QTproj/porsche.jpg");

```



```

int w = ui->label_16->width();
int h = ui->label_16->height();
ui->label_16->setPixmap(pix.scaled(w,h,Qt::KeepAspectRatio));
QPixmap pixx("/Users/andrey/Documents/QTproj/Porsche-Logo.png");
int ww = ui->label_17->width();
int hh = ui->label_17->height();
ui->label_17->setPixmap(pixx.scaled(ww,hh,Qt::KeepAspectRatio));

// регистрация слушателя
connect(&info, SIGNAL(notifyObservers()), this, SLOT(update())); //включаем сигнал для
наблюдателя включающийся при изменении данных

connect(ui->btnCalc, SIGNAL(pressed()), this, SLOT(btnCalcPressed())); //включаем сигналы
включающиеся при нажатии кнопок

connect(ui->btnUndo, SIGNAL(pressed()), this, SLOT(btnUndoPressed()));
connect(ui->btnAddUser, &QPushButton::clicked, this, &Widget::btnUserPressed);
connect(ui->btnIssue, &QPushButton::clicked, this, &Widget::btnIssuePressed);
connect(ui->btnRefund, &QPushButton::clicked, this, &Widget::btnSetRefund);

//connect(ui->moveToUsedButton, &QPushButton::clicked, this, &Widget::moveToUsedTable);
//connect(ui->moveToTableButton, &QPushButton::clicked, this, &Widget::moveToTableView);

//Widget обновляет свое состояние и затем уведомляет о этом изменении, вызывая метод
update(). В ответ на этот вызов метода, наблюдатели реагируют на изменение.
}

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

//public slots
void Widget::update(){
    auto value = info.getActualData(); //получаем актуальную информацию
    if(value != nullptr){ //если значение не пустое
        fillForm(value); //выводим на форму
    }

    //update btnUndo state
    ui->btnUndo->setEnabled(info.hasCars());

    //seting value to null
    value=nullptr;

```

```

}

//private slots

void Widget::btnCalcPressed() {           //функции добавляют новый автомобиль

    auto value=processForm();//создаем объект класса

    showCost(value);//вычисляем стоимость и выводим ее

    controller.addToTableView(value,ui->tableView);

    info.add(value);//добавляем объект в коллекцию предыдущих запросов

    carsinfo.add(value);

    ui->btnUndo->setEnabled(info.hasCars());

    //seting value to null

    value=nullptr;

}

void Widget::btnUndoPressed(){

    info.undo();//запрос на получение информации о прошлом запросе

    ui->cost->setText("0");//стоимость 0

}

void Widget::btnUserPressed() {           //функции добавляют нового пользователя

    auto value=processClientForm();

    controller.addToClientTable(value, ui->userTable);

    //addToUserTable(value,ui->userTable);

    clientinfo.add(value);

    value=nullptr;

}

void Widget::btnIssuePressed(){

    //extraditioninfo.add(addToTableRefund(ui->refundTable));

    rentinfo.add(controller.addToTableRefund(ui->refundTable,ui->userTable,ui->tableView,carsinfo,ui->issueDate)); //добавляет информацию об аренде в таблицу возвратов (refundTable)

}

void Widget::btnSetRefund(){

    //setRefundData(ui->refundTable);

    controller.setRefundData(ui->refundTable,ui->tableView,rentinfo,ui->refundDate,carsinfo);//вносит изменения в таблицу возвратов (refundtable), изменяя данные о дате возврата и

    //обновляя статус автомобиля в таблице авто (cartable). В
    конце функции обновляется модель данных в обеих таблицах.

}

```

```

//private
car *Widget::processForm() { //берем данные с формы и создаем новый объект класса
    int age = ui->Age->text().toInt();
    int ID = ui->Number->text().toInt();
    QString Colour = ui->Colour->text();
    car::CarType type = static_cast<car::CarType>(ui->CarType->currentIndex());
    QString Brand = ui->Brand->text();
    return new car(age, ID, Colour, type, Brand);
}

client *Widget::processClientForm(){
    int Passport = ui->passport->text().toInt();
    int PhoneNumber = ui->phonenummer->text().toInt();
    QString FIOclient = ui->FIOclient->text();
    QString EMail = ui->EMail->text();
    return new client (Passport,PhoneNumber,FIOclient,EMail);
}

void Widget::fillForm(car *value) { //заполняем форму актуальной информацией
    QString str=value->getBrand();
    ui->Brand->setText(str);

    str=QString::number(value->getAge());
    ui->Age->setText(str);

    if (value->getType() == car::CarType::STANDARD) {
        ui->CarType->setCurrentIndex(0);
    } else if (value->getType() == car::CarType::COMFORT) {
        ui->CarType->setCurrentIndex(1);
    } else if (value->getType() == car::CarType::LUXURY) {
        ui->CarType->setCurrentIndex(2);
    } else if (value->getType() == car::CarType::ELECTRIC) {
        ui->CarType->setCurrentIndex(3);
    }

    str=value->getColour();
    ui->Colour->setText(str);
}

```

```

    str=QString::number(value->getNumber());
    ui->Number->setText(str);
}

QString Widget::showCost(car *value){
    CalculationFacade cur;//создаем объект фасада вычисления
    int rating=cur.getCost(value);//получаем стоимость от фасада
    QString str=QString::number(rating);//переводим тип данных стоимости из str в QString
    ui->cost->setText(str);//выводим стоимость на форму
    return str;
}

```


5. Пример выполнения программы

Вид программы при открытии:

The screenshot shows a Qt-based application window titled "Захаров ЛР8". The interface is designed for a Porsche car rental system. It features a header with the Porsche logo and name. The main area is divided into several sections:

- Car Details Form:** Includes fields for Model (Carrera GT), Serial Number (12034), Equipment (Базовая), Color (Moonwalk Grey), Year of Production (2006), and First Payment (0). There are also fields for Passport Data (1234567891), FIO (Калинин С.К.), Email (mbam@gmail.com), and Phone (12345).
- Customer Management:** A "Добавить клиента" button and a "Последний запрос" button.
- Rental Dates:** Fields for "Дата выдачи" (22.05.2023, 10:00) and "Дата возврата" (22.05.2023, 15:00).
- Buttons:** "Добавить", "Выдать", and "Вернуть".
- Car List:** A section titled "Список доступных Автомобилей" with a large empty box for displaying available cars.
- Customer Information and History:** Two empty boxes labeled "Информация о покупателе" and "История операций".
- Image:** A photograph of a Porsche dealership building with several cars parked in front.

Захаров ЛРВ



PORSCHE

Список доступных Автомобилей

	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12034	Moonwalk ...	2006	100300	В наличии
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии

Информация о покупателе

История операций

Модель: Carrera GT

Серийный номер: 12031

Комплектация: Комфорт

Цвет: Moonwalk Grey

Год производства: 2006

первый взнос: 140420

Добавить

Последний запрос

Паспортные данные: 1234567891

ФИО: Калинин С.К.

электронная почта: mbam@gmail.com

Телефон: 12345


Дата выдачи 22.05.2023, 10:00

Дата возврата 22.05.2023, 15:00


Выдать

Вернуть

Добавить клиента



Захаров ЛРВ



PORSCHE

Список доступных Автомобилей

Модель: Carrera GT

Серийный номер: 12034

Комплектация: Базовая

Цвет: Moonwalk Grey

Год производства: 2006

первый взнос: 0

Добавить

Последний запрос

Паспортные данные: 1234567891

ФИО: Калинин С.К.

электронная почта: mbam@gmail.com

Телефон: 12345

Добавить клиента

Дата выдачи: 22.05.2023, 10:00

Дата возврата: 22.05.2023, 15:00


Выдать

Вернуть

	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12034	Moonwalk ...	2006	100300	В наличии
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии


Информация о покупателе

История операций



Добавили ещё автомобилей и клиентов

Захаров ЛРВ



PORSCHE

Список доступных Автомобилей

	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12034	Moonwalk ...	2006	100300	В наличии
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии
3	Carrera GT	Люкс	12032	Moonwalk ...	2006	180540	В наличии
4	Carrera GT	Электро	12033	Moonwalk ...	2006	160480	В наличии

Информация о покупателе

Паспорт	ФИО	EMail	Номер	
1	1234567891	Калинин С.К.	bimbibam...	12345
2	1234567892	Калинин С.К.	bimbibam...	12345
3	1234567893	Калинин С.К.	bimbibam...	12345
4	1234567894	Калинин С.К.	bimbibam...	12345

История операций

Модель: Carrera GT

Паспортные данные: 1234567894

Серийный номер: 12033

ФИО: Калинин С.К.

Комплектация: Электро

электронная почта: mbam@gmail.com

Цвет: Moonwalk Grey

Телефон: 12345

Год производства: 2006

первый взнос: 160480

Добавить клиента

Дата выдачи: 22.05.2023, 10:00


Дата возврата: 22.05.2023, 15:00

Добавить

Последний запрос


Выдать

Вернуть



Первому покупателю выдаём первый автомобиль – появляется данная операция и статус первого автомобиля меняется с «В наличии» на «Выдана»

Захаров ЛРВ



PORSCHE

Список доступных Автомобилей

	Название	Комплектация	Номер	Цвет	Год выпуска	Взнос	Наличие
1	Carrera GT	Стандарт	12034	Moonwalk ...	2006	100300	Выдана
2	Carrera GT	Комфорт	12031	Moonwalk ...	2006	140420	В наличии
3	Carrera GT	Люкс	12032	Moonwalk ...	2006	180540	В наличии
4	Carrera GT	Электро	12033	Moonwalk ...	2006	160480	В наличии

Информация о покупателе

Паспорт	ФИО	EMail	Номер	
1	1234567891	Калинин С.К.	bimbibam...	12345
2	1234567892	Калинин С.К.	bimbibam...	12345
3	1234567893	Калинин С.К.	bimbibam...	12345
4	1234567894	Калинин С.К.	bimbibam...	12345

История операций

Паспорт	Серийный номер	Дата выдачи	Дата возврата
1	1234567891	12034	22.05.2023, ...

Модель: Carrera GT

Паспортные данные: 1234567894

Серийный номер: 12033

ФИО: Калинин С.К.

Комплектация: Электро

электронная почта: mbam@gmail.com

Цвет: Moonwalk Grey

Телефон: 12345

Год производства: 2006

первый взнос: 160480

Добавить клиента

Дата выдачи: 22.05.2023, 10:00


Дата возврата: 22.05.2023, 15:00

Добавить

Последний запрос


Выдать

Вернуть



Пробуем выдать первый автомобиль(выданный) второму клиенту – операции нет – выдать невозможно

Захаров ЛР8



PORSCHE

Список доступных Автомобилей

Модель: Carrera GT

Паспортные данные: 1234567894

Серийный номер: 12033

ФИО: Калинин С.К.

Комплектация: Электро

электронная почта: mbam@gmail.com

Цвет: Moonwalk Grey

Телефон: 12345

Год производства: 2006

первый взнос: 160480

Добавить клиента

Дата выдачи: 22.05.2023, 10:00

Дата возврата: 22.05.2023, 15:00

Добавить

Последний запрос

Выдать

Вернуть

1

Carrera GT

Стандарт

12034

Moonwalk ...

2006

100300

Выдана

2

Carrera GT

Комфорт

12031

Moonwalk ...

2006

140420

В наличии

3

Carrera GT

Люкс

12032

Moonwalk ...

2006

180540

В наличии

4

Carrera GT

Электро

12033

Moonwalk ...

2006

160480

В наличии

Информация о покупателе

История операций

Паспорт

ФИО

E-Mail

Номер

1

1234567891

Калинин С.К.

bimbibam...

12345

2

1234567892

Калинин С.К.

bimbibam...

12345

3

1234567893

Калинин С.К.

bimbibam...

12345

4

1234567894

Калинин С.К.

bimbibam...

12345

Паспорт

Серийный номер

Дата выдачи


Дата возврата

1

1234567891


12034

22.05.2023, ...



Выдаём второму клиенту вторую машину – появляется операция и меняется статус второй машины, в это же время возвращаем первую машину – заносится дата возврата в операциях и меняется статус первой машины на доступный

Захаров ЛР8



PORSCHE

Список доступных Автомобилей

Модель: Carrera GT

Паспортные данные: 1234567894

Серийный номер: 12033

ФИО: Калинин С.К.

Комплектация: Электро

электронная почта: mbam@gmail.com

Цвет: Moonwalk Grey

Телефон: 12345

Год производства: 2006

первый взнос: 160480

Добавить клиента

Дата выдачи: 22.05.2023, 10:00

Дата возврата: 22.05.2023, 15:00

Добавить

Последний запрос

Выдать

Вернуть

1

Carrera GT

Стандарт

12034

Moonwalk ...

2006

100300

В наличии

2

Carrera GT

Комфорт

12031

Moonwalk ...

2006

140420

Выдана

3

Carrera GT

Люкс

12032

Moonwalk ...

2006

180540

В наличии

4

Carrera GT

Электро

12033

Moonwalk ...

2006

160480

В наличии

Информация о покупателе

История операций

Паспорт

ФИО

E-Mail

Номер

1

1234567891

Калинин С.К.

bimbibam...

12345

2

1234567892

Калинин С.К.

bimbibam...

12345

3

1234567893

Калинин С.К.

bimbibam...

12345

4

1234567894

Калинин С.К.

bimbibam...

12345

Паспорт

Серийный номер

Дата выдачи

Дата возврата

1

1234567891

12034

22.05.2023, ...


22.05.2023, ...

2

1234567892

12031

22.05.2023, ...



6. Анализ результатов и выводы

В результате выполнения лабораторной работы были изучены принципы применения на практике паттернов проектирования.