

ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

Старший преподаватель

должность, уч. степень, звание

подпись, дата

Е.О. Шумова

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6

**«Структурные и поведенческие шаблоны проектирования»**

по курсу: ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4134к

подпись, дата

Костяков Н.А.

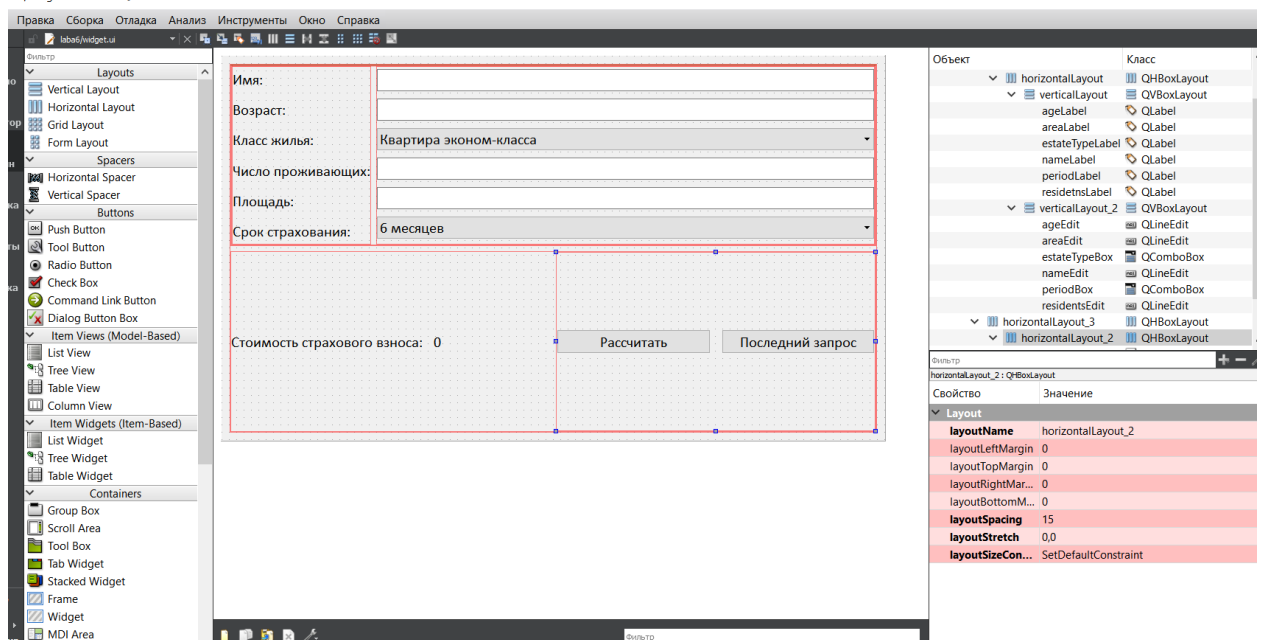
инициалы, фамилия

Санкт-Петербург 2022

## Цель работы

Изучить принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

## Вид исходной формы



## Диаграммы классов для паттернов проектирования

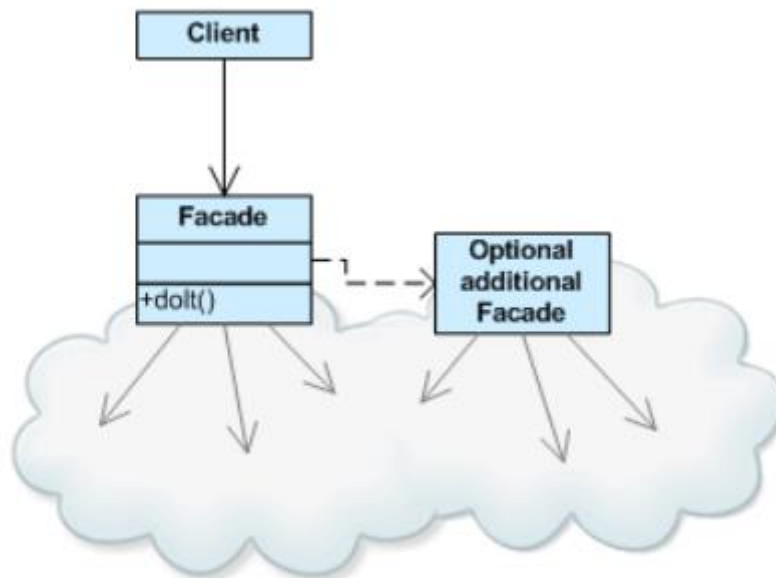


Рис. 10. Диаграмма классов паттерна Facade

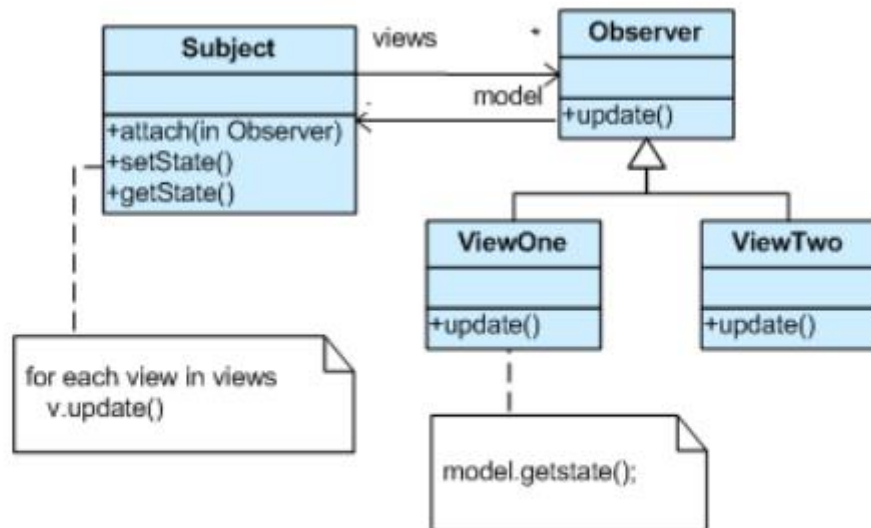


Рис. 11. Диаграмма классов шаблона Observer

## Листинг программы

```

apartmentcalc.cpp
#include "apartmentcalc.h"

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

}

int apartmentCalc::getCost(estate *value){
    return (value->getAge() + value->getArea() + value->getMonths() + value->getResidents())
    * 770;
}

apartmentcalc.h
#ifndef APARTMENTCALC_H
#define APARTMENTCALC_H

#include <QObject>
#include <estate.h>

class apartmentCalc : public QObject
{
    Q_OBJECT
public:
    explicit apartmentCalc(QObject *parent = nullptr);
    static int getCost(estate *value);
};

#endif // APARTMENTCALC_H
  
```

calculationfacade.cpp

```
#include "calculationfacade.h"
```

```
calculationFacade::calculationFacade(QObject *parent)
: QObject{parent}
{
```

```
}
```

```
// исходя из типа вызывает, метод у нужного класса
```

```
int calculationFacade::getCost(estate *value){
    double cost;
    switch(value->getType()){
        case estate::EstateType::ECONOM:
            cost = apartmentCalc::getCost(value);
            break;
        case estate::EstateType::LUXURIOUS:
            cost = luxuriousCalc::getCost(value);
            break;
        case estate::EstateType::TOWN_HOUSE:
            cost = townhouseCalc::getCost(value);
            break;
        case estate::EstateType::COTTAGE:
            cost = cottageCalc::getCost(value);
            break;
        default:
            cost = -1;
            break;
    }
    return cost;
}
```

calculationfacade.h

```
#ifndef CALCULATIONFACADE_H
```

```
#define CALCULATIONFACADE_H
```

```
#include <QObject>
```

```
#include <estate.h>
```

```
#include <apartmentcalc.h>
```

```
#include <luxuriouscalc.h>
```

```
#include <townhousecalc.h>
```

```
#include <cottagecalc.h>
```

```
class calculationFacade : public QObject
```

```
{
```

```
    Q_OBJECT
```

```
public:
```

```
    explicit calculationFacade(QObject *parent = nullptr);
```

```
    static int getCost(estate *value);
```

```
};
```

```
#endif // CALCULATIONFACADE_H
```

```
cottagecalc.cpp
```

```
#include "cottagecalc.h"
```

```
cottageCalc::cottageCalc(QObject *parent)
```

```
: QObject{parent}
```

```
{
```

```
}
```

```
int cottageCalc::getCost(estate *value){
```

```
    return (value->getAge() + value->getArea() + value->getMonths() + value->getResidents())
```

```
* 2700;
```

```
}
```

```
cottagecalc.h
```

```
#ifndef COTTAGECALC_H
```

```
#define COTTAGECALC_H
```

```
#include <QObject>
```

```
#include <estate.h>
```

```
class cottageCalc : public QObject
```

```
{
```

```
    Q_OBJECT
```

```
public:
```

```
    explicit cottageCalc(QObject *parent = nullptr);
```

```
    static int getCost(estate *value);
```

```
};
```

```
#endif // COTTAGECALC_H
```

```
estate.cpp
```

```
#include "estate.h"
```

```
#include <widget.h>
```

```
estate::estate(QObject *parent)
```

```
: QObject{parent}
```

```
{
```

```
}
```

```
estate::estate(const QString owner, const int age, const int type,
```

```
               const int residents, const double area, const QString months){
```

```
    if (owner == "" || age == 0 || residents == 0 || area == 0)
```

```
        throw Exception("Заполните все поля формы."); // проверка на то что все поля  
        заполнены
```

```
    this->age = age;
```

```
    this->area = area;
```

```
    this->residents = residents;
```

```

    this->months = months.split(" ")[0].toInt();
    this->owner = owner;
    this->type = static_cast<EstateType>(type); // присваиваем тип
}

```

```

estate::EstateType estate::getType() const{
    return this->type;
}

```

```

int estate::getAge() const{
    return this->age;
}

```

```

double estate::getArea() const{
    return this->area;
}

```

```

int estate::getMonths() const{
    return this->months;
}

```

```

int estate::getResidents() const{
    return this->residents;
}

```

```

QString estate::getOwner() const{
    return this->owner;
}

```

```

estate.h
#ifndef ESTATE_H
#define ESTATE_H

```

```

#include <QObject>

```

```

class estate : public QObject
{
    Q_OBJECT

```

```

public:

```

```

    explicit estate(QObject *parent = nullptr);

```

```

    enum EstateType{
        ECONOM,
        LUXURIOUS,
        TOWN_HOUSE,
        COTTAGE
    };

```

```

    estate(const QString owner, const int age, const int type,
           const int residents, const double area, const QString months);

```

```

    EstateType getType() const; //задание 1

```

```

    int getAge() const;

```

```

    int getMonths() const;

```

```

    double getArea() const;

```

```

    int getResidents() const;
    QString getOwner() const;

```

```

private:

```

```

    int age, residents, months;
    double area;
    EstateType type;
    QString owner;
};

```

```

#endif // ESTATE_H

```

```

exception.h

```

```

#ifndef EXCEPTION_H
#define EXCEPTION_H

```

```

#include <QException>
#include <QMessageBox>

```

```

class Exception : public QException

```

```

{

```

```

public:

```

```

    Exception(QString const &text = " ") noexcept : msg(text) {}
    Exception(const Exception &err) { this->msg = err.msg; }
    ~Exception() override {}
    void raise() const override { throw *this; }
    Exception *clone() const override { return new Exception(*this); }
    const char *what() const noexcept override { return this->msg.toStdString().c_str(); }

```

```

private:

```

```

    QString msg;
};

```

```

#endif // EXCEPTION_H

```

```

luxuriouscalc.cpp

```

```

#include "luxuriouscalc.h"

```

```

luxuriousCalc::luxuriousCalc(QObject *parent)

```

```

    : QObject{parent}

```

```

{

```

```

}

```

```

int luxuriousCalc::getCost(estate *value){

```

```

    return (value->getAge() + value->getArea() + value->getMonths() + value->getResidents())
    * 1900;
}

```

```

luxuriouscalc.h

```

```

#ifndef LUXURIOUSCALC_H

```

```

#define LUXURIOUSCALC_H

```

```

#include <QObject>
#include <estate.h>

class luxuriousCalc : public QObject
{
    Q_OBJECT
public:
    explicit luxuriousCalc(QObject *parent = nullptr);
    static int getCost(estate *value);

};

#endif // LUXURIOUSCALC_H

```

main.cpp

```

#include "widget.h"

```

```

#include <QApplication>

```

```

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

```

states.cpp

```

#include "states.h"

```

*//класс, который обрабатывает списки, и реализует кнопку "вернуться назад"*

```

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

```

```

states::~states(){ //деструктор

```

```

    if (actualData){
        delete actualData;
        actualData = nullptr;
    }

```

```

    qDeleteAll(array);
    array.clear();
}

```

```

bool states::hasStates(){ // проверка на существование
    return !(array.isEmpty());
}

```

```

estate* states::getActualData(){ // возврат элемента
    return actualData;
}

```



```

}

void states::add(estate* value){ // добавление
    array.append(value);
}

```

```

void states::undo(){
    if (hasStates()){
        array.pop_back();
        actualData = array.last();
        emit notifyObservers();
    }
    else actualData = nullptr;
}

```

```

int states::getSize(){
    return array.size();
}

```

states.h

```

#ifndef STATES_H
#define STATES_H

```

```

#include <QObject>
#include <estate.h>

```

**class states** : **public** QObject *// класс, который обрабатывает списки, и реализует кнопку "вернуться назад"*

```

{
    Q_OBJECT
public:
    explicit states(QObject *parent = nullptr);
    ~states();

```

```

    void undo();
    bool hasStates();
    estate *getActualData();
    void add(estate *value);
    int getSize();

```

signals:

```

    void notifyObservers();

```

private:

```

    QList<estate*> array;
    estate *actualData;
};

```

```

#endif // STATES_H

```

townhousecalc.cpp

```

#include "townhousecalc.h"

```

```

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

}

int townhouseCalc::getCost(estate *value){
    return (value->getAge() + value->getArea() + value->getMonths() + value->getResidents())
    * 2500;
}

```

```

townhousecalc.h
#ifndef TOWNHOUSECALC_H
#define TOWNHOUSECALC_H

```

```

#include <QObject>
#include <estate.h>

class townhouseCalc : public QObject
{
    Q_OBJECT
public:
    explicit townhouseCalc(QObject *parent = nullptr);
    static int getCost(estate *value);

};

```

```

#endif // TOWNHOUSECALC_H

```

```

widget.cpp
#include "widget.h"
#include "ui_widget.h"

```

```

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget),
    forIntValidator(QRegularExpression("^([0-9])+")),
    forDoubleValidator(QRegularExpression("^([0-9]*[.]?[0-9])+")),
    forOwnerValidator(QRegularExpression("^([A-Я][a-я]+)\\s([A-Я][a-яA-Я-]+)$")),
    info(this)
{
    ui->setupUi(this);
    ui->undoButton->setEnabled(false);
    ui->ageEdit->setValidator(&forIntValidator);
    ui->residentsEdit->setValidator(&forIntValidator);
    ui->areaEdit->setValidator(&forDoubleValidator);
    ui->nameEdit->setValidator(&forOwnerValidator);

    connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));
    connect(ui->calcButton, SIGNAL(pressed()), this, SLOT(calcPressed()));
    connect(ui->undoButton, SIGNAL(pressed()), this, SLOT(undoPressed()));
}

```

```

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

void Widget::update() {
    auto value = info.getActualData();
    if (value != nullptr) fillForm(value);
    ui->undoButton->setEnabled(info.hasStates());
    value = nullptr;
}

void Widget::calcPressed() {
    try {
        auto value = processForm();
        showCost(value);
        info.add(value);
        ui->undoButton->setEnabled(true);
        value = nullptr;
    }
    catch(const Exception &error){
        QMessageBox msg;
        msg.setWindowTitle("Ошибка!");
        msg.setFixedSize(500,400);
        msg.setText(error.what());
        msg.exec();
        return;
    }
}

void Widget::undoPressed() { //
    if (info.getSize() > 1) info.undo();
    else return;
}

estate *Widget::processForm() {
    return new estate(ui->nameEdit->text(), ui->ageEdit->text().toInt(), ui->estateTypeBox->currentIndex(),
        ui->residentsEdit->text().toInt(), ui->areaEdit->text().toDouble(), ui->periodBox->currentText());
}

void Widget::fillForm(estate *value) {
    ui->nameEdit->setText(info.getActualData()->getOwner());
    ui->ageEdit->setText(QString::number(info.getActualData()->getAge()));
    ui->residentsEdit->setText(QString::number(info.getActualData()->getResidents()));
    ui->periodBox->setCurrentIndex((info.getActualData()->getMonths() / 6) - 1);
    ui->areaEdit->setText(QString::number(info.getActualData()->getArea()));
    switch (info.getActualData()->getType()) {
        case estate::EstateType::ECONOM:
            ui->estateTypeBox->setCurrentIndex(0);
    }
}

```

```

        break;
    case estate::EstateType::LUXURIOUS:
        ui->estateTypeBox->setCurrentIndex(1);
        break;
    case estate::EstateType::TOWN_HOUSE:
        ui->estateTypeBox->setCurrentIndex(2);
        break;
    case estate::EstateType::COTTAGE:
        ui->estateTypeBox->setCurrentIndex(3);
        break;
    }
    showCost(value);
}

void Widget::showCost(estate *value)
{
    ui->costLabel->setText("Стоимость страхового взноса: " +
    QString::number(calculationFacade::getCost(value)));
}

```

widget.h

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <states.h>
#include <estate.h>
#include <calculationFacade.h>
#include <exception.h>
#include <QRegularExpressionValidator>
#include <QRegularExpression>

```

```

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

```

```

class Widget : public QWidget
{
    Q_OBJECT

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

public slots:
    void update();

private slots:
    void calcPressed();
    void undoPressed();

```

```

private:

```

```

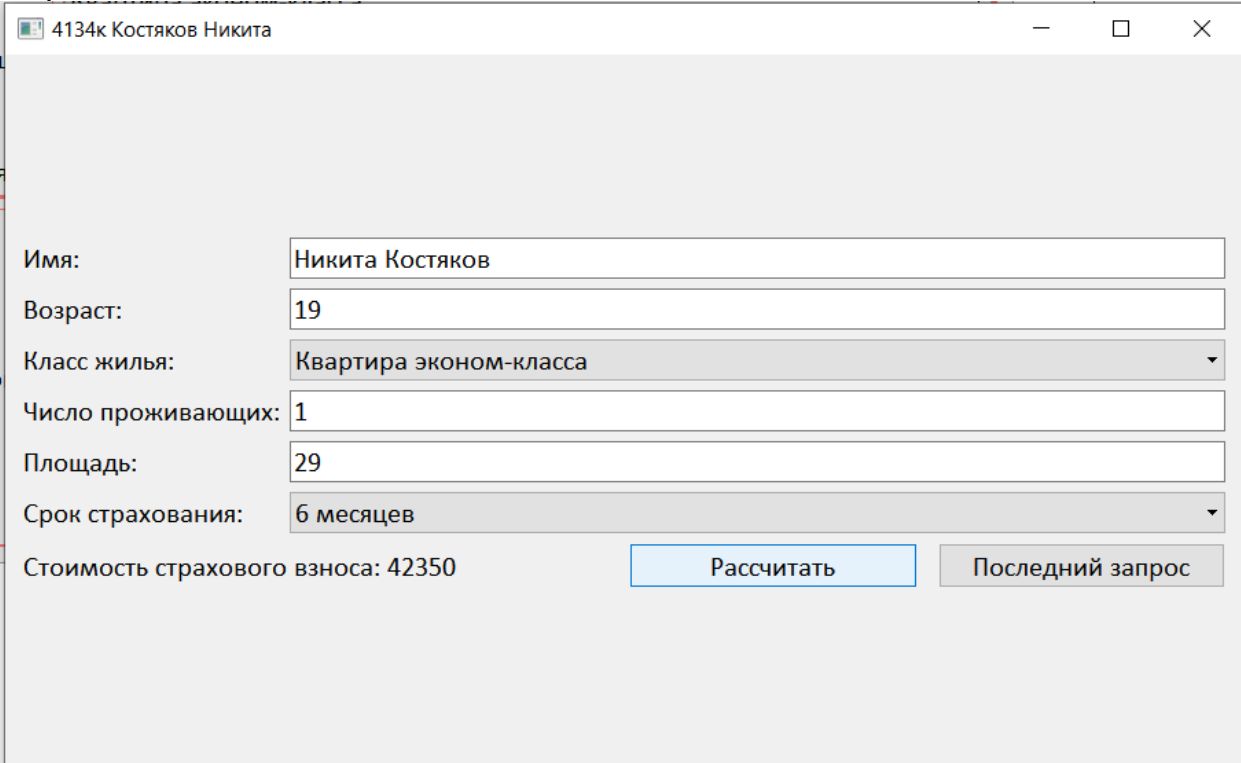
estate *processForm();
void fillForm(estate *value);
void showCost(estate *value);

private:
    Ui::Widget *ui;
    QRegularExpressionValidator forIntValidator, forDoubleValidator,
                                forOwnerValidator;
    states info;
};
#endif // WIDGET_H

```

## Результат работы программы

Запрос 1



4134к Костяков Никита

Имя:	<input type="text" value="Никита Костяков"/>
Возраст:	<input type="text" value="19"/>
Класс жилья:	<input type="text" value="Квартира эконом-класса"/>
Число проживающих:	<input type="text" value="1"/>
Площадь:	<input type="text" value="29"/>
Срок страхования:	<input type="text" value="6 месяцев"/>

Стоимость страхового взноса: 42350

Запрос 2

4134к Костяков Никита

Имя: Никита

Возраст: 20

Класс жилья: Элитная квартира

Число проживающих: 1

Площадь: 38

Срок страхования: 6 месяцев

Стоимость страхового взноса: 123500

Рассчитать

Последний запрос

### Запрос 3

4134к Костяков Никита

Имя: Андрей

Возраст: 22

Класс жилья: Таун-хаус

Число проживающих: 2

Площадь: 144

Срок страхования: 6 месяцев

Стоимость страхового взноса: 435000

Рассчитать

Последний запрос

Кнопка последний запрос откатывает форму на предыдущий запрос

1 Нажатие

4134к Костяков Никита

Имя: Никита

Возраст: 20

Класс жилья: Элитная квартира

Число проживающих: 1

Площадь: 38

Срок страхования: 6 месяцев

Стоимость страхового взноса: 123500

Рассчитать

Последний запрос

2 нажатие

4134к Костяков Никита

Имя: Никита Костяков

Возраст: 19

Класс жилья: Квартира эконом-класса

Число проживающих: 1

Площадь: 29

Срок страхования: 6 месяцев

Стоимость страхового взноса: 42350

Рассчитать

Последний запрос

## Выводы

Я изучил принципы построения приложений с графическим интерфейсом