ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподователь		Е. О. Шумова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

«Порождающие шаблоны проектирования»

по курсу: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

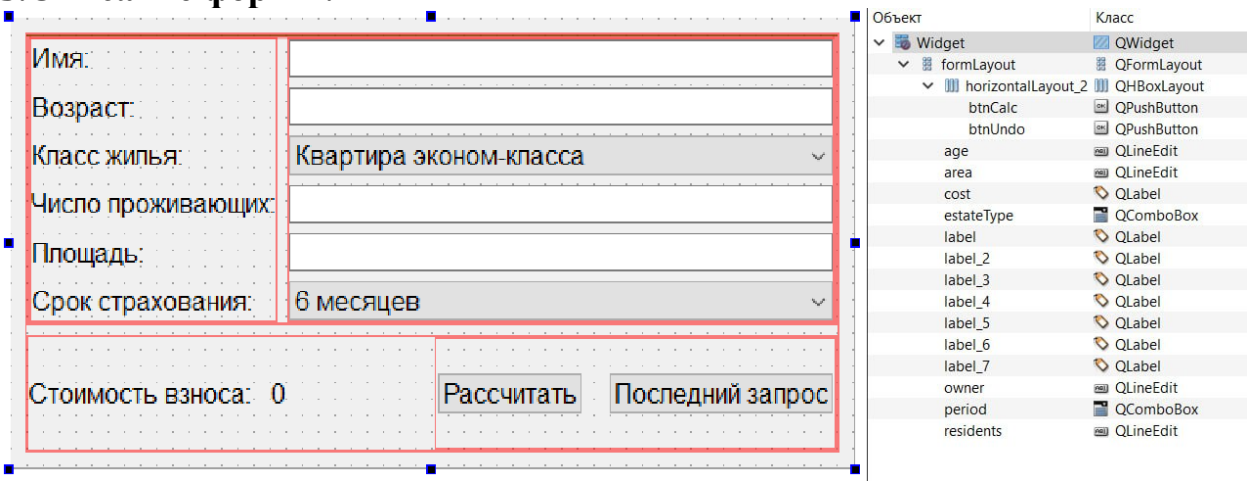
РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4134К		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

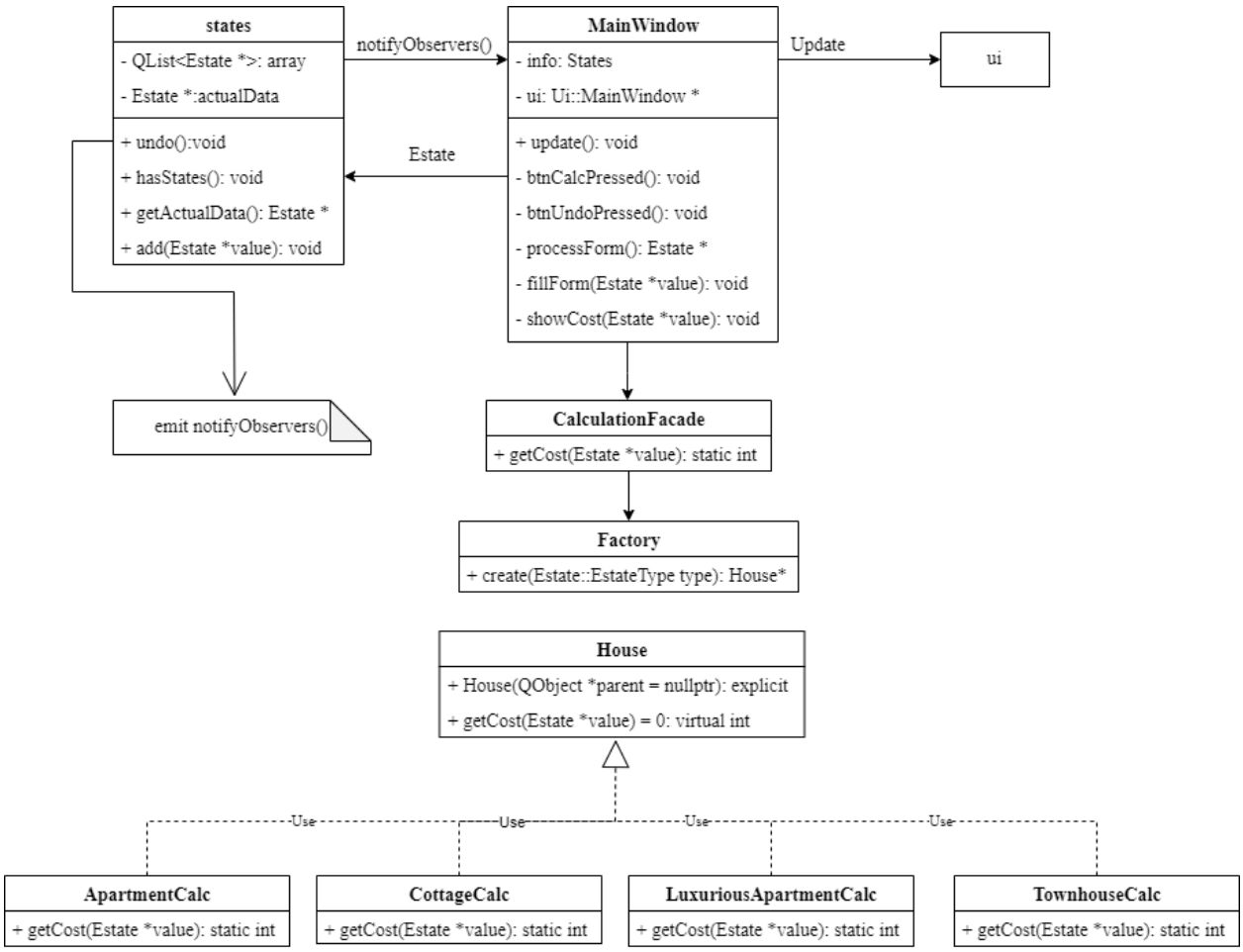
1.Цель работы:
Изучить принципы построения приложений с графическим интерфейсом, использую библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

2.Задачи работы:
Выполнить все требования к лабораторной работе, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

3.Описание формы:


- Виджет owner дает пользователю ввести имя (должно начинаться с заглавной буквы, не допускаются символы)
- Виджеты age, residents и area дают ввести возраст, число проживающих и площадь соответственно (только числа)
- Виджеты estateType и period позволяют пользователю выбрать класс жилья и срок страхования соответственно.
- Виджет cost отображает в себе стоимость взноса.
- Виджет btnCalc расчитывает стоимость взноса и отображает ее в cost.
- Виджет btnUndo позволяет вернуться к предыдущему запросу до тех пор, пока такой существует, иначе кнопка становиться нерабочей.

4.Диаграммы классов:



5.ЛИСТИНГ КОДА:

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
```

Main.cpp:

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

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

    return a.exec();
}
```

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);

    // регистрация слушателя
    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;
}
//расчет
void Widget::calcPressed(){
    try {
    auto value = processForm();
    showCost(value);
    info.add(value);
    ui->undoButton->setEnabled(true);
    value = nullptr;
    //update();
    }
    catch(const myException &error){
        QMessageBox msg;
        msg.setWindowTitle("Ошибка!");
        msg.setFixedSize(400,400);

        msg.setText(error.what());
        msg.setText("Заполните поля!          ");
        msg.exec();
        return;
    }
}
//предыдущий запрос
void Widget::undoPressed(){
    if (info.getSize() > 1) info.undo();
    else {
        //qInfo() << info.hasStates();
        ui->undoButton->setEnabled(false);
        return;
    }
}
//Обработка формы и создание нового Estate
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());
}

//Заполнение формы
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);
}
```

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

Estate.h:

```
#ifndef ESTATE_H
#define ESTATE_H

#include <QObject>

class Estate : public QObject
{
    Q_OBJECT
public:
    enum EstateType {
        ECONOM,
        LUXURIOUS,
        TOWN_HOUSE,
        COTTAGE
    };
    explicit Estate(int inputAge, int inputArea,int inputResidents, int
inputMonths, EstateType inputEstateType,QString inputOwner,QObject *parent
= nullptr);
    EstateType getType();
    int getMonths();
    int getArea();
    int getResidents();
    QString getName();
    int getAge();
private:
    int age;
    int area;
    int residents;
    int months;
    EstateType type;
    QString owner;
};

#endif // ESTATE_H
```

Estate.cpp:

```
#include "estate.h"

Estate::Estate(int inputAge, int inputArea,int inputResidents, int
inputMonths, EstateType inputEstateType,QString inputOwner,QObject *parent)
    : QObject{parent}
{
    age=inputAge;
    area=inputArea;
    residents=inputResidents;
    months=inputMonths;
    type=inputEstateType;
    owner=inputOwner;
}

Estate::getArea(){
    return area;
}

Estate::getMonths(){
```

```
    return months;
}

Estate::getResidents(){
    return residents;
}

Estate::EstateType Estate::getType(){
    return type;
}

QString Estate::getName(){
    return owner;
}

int Estate::getAge(){
    return age;
}
```

States.h:

```
#ifndef STATES_H
#define STATES_H

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

class States : public QObject
{
    Q_OBJECT
public:
    explicit States(QObject *parent = nullptr);
    ~States();
    void undo();
    bool hasStates();
    Estate *getActualData();
    void add(Estate *value);
private:
    QList<Estate *> array;
    Estate *actualData;
signals:
    void notifyObservers();
};

#endif // STATES_H
```

States.cpp:

```
#include "states.h"

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

States::~States()
{
    //delete: actualData
    if(actualData){
        delete actualData;
        actualData=nullptr;
    }
    //delete and clear array
    array.clear();
```

```
    qDeleteAll(array);

}

bool States::hasStates(){
    return !array.empty();
}

    Estate *States::getActualData(){
        //return array.takeLast();
        return array.back();
        //return array.takeAt(array.size()-1);
    }

void States::add(Estate *value){
    array.append(value);
}

void States::undo(){
    if(!hasStates()||(array.size()==1)){
        actualData=nullptr;
    }
    else {
        actualData=getActualData();
        array.removeLast();
        emit notifyObservers();
    }
}
```

CalculationFacade.h:

```
#ifndef CALCULATIONFACADE_H
#define CALCULATIONFACADE_H

#include <QObject>
#include <apartmentfactory.h>
#include <luxuriousfactory.h>
#include <cottagefactory.h>
#include <townhousefactory.h>

class calculationFacade : public QObject
{
    Q_OBJECT
public:
    explicit calculationFacade(QObject *parent = nullptr);
    static double getCost(estate *value);
    ~calculationFacade();
private:
    static apartmentFactory* apartment_factory;
    static luxuriousFactory* luxurious_factory;
    static cottageFactory* cottage_factory;
    static townhouseFactory* townhouse_factory;
};

#endif // CALCULATIONFACADE_H
// статические объекты для избежания утечек памяти
```

CalculationFacade.cpp:

```
#include "calculationfacade.h"

apartmentFactory* calculationFacade::apartment_factory =  new
apartmentFactory;
```

```
luxuriousFactory* calculationFacade::luxurious_factory =  new
luxuriousFactory;
cottageFactory* calculationFacade::cottage_factory =  new cottageFactory;
townhouseFactory* calculationFacade::townhouse_factory =  new
townhouseFactory;

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

}
calculationFacade::~calculationFacade(){

}

double calculationFacade::getCost(estate *value){
    abstractCalc* house;
    switch(value->getType()){
    case estate::EstateType::ECONOM:{
        house = apartment_factory->createCalc();
        break;
    }
    case estate::EstateType::LUXURIOUS:{
        house = luxurious_factory->createCalc();
        break;
    }
    case estate::EstateType::TOWN_HOUSE:{
        house = townhouse_factory->createCalc();
        break;
    }
    case estate::EstateType::COTTAGE:{
        house = cottage_factory->createCalc();
        break;
    }
    default:{
        return 0;
        break;
    }
 }
    return house->getCost(value);
}

// абстрактный дом + нужный тип данных для вычислений, для подключения
нужной фабрики
```

apartmentcalc.h:

```
#ifndef APARTMENTCALC_H
#define APARTMENTCALC_H

#include <abstractcalc.h>

class apartmentCalc : public abstractCalc
{
public:
    virtual double getCost(estate* value);
};

#endif // APARTMENTCALC_H
```

apartmentcalc.cpp:

```
#include "apartmentcalc.h"

double apartmentCalc::getCost(estate *value){
```

```
    return (value->getAge() + value->getArea() + value->getMonths() +
value->getResidents()) * 1000;
}
```

cottagecalc.h:

```
#ifndef COTTAGECALC_H
#define COTTAGECALC_H

#include <abstractcalc.h>

class cottageCalc : public abstractCalc
{
public:
    virtual double getCost(estate* value);

};

#endif // COTTAGECALC_H
```

cottagecalc.cpp:

```
#include "cottagecalc.h"

double cottageCalc::getCost(estate *value){
    return (value->getAge() + value->getArea() + value->getMonths() +
value->getResidents()) * 3000;
}
```

luxuriouscalc.h:

```
#ifndef LUXURIOUSCALC_H
#define LUXURIOUSCALC_H

#include <abstractcalc.h>

class luxuriousCalc : public abstractCalc
{
public:
    virtual double getCost(estate* value);

};

#endif // LUXURIOUSCALC_H
```

luxuriouscalc.cpp:

```
#include "luxuriouscalc.h"

double luxuriousCalc::getCost(estate *value){
   return (value->getAge() + value->getArea() + value->getMonths() + value-
>getResidents()) * 1500;
}
```

townhousecalc.h:

```
#ifndef TOWNHOUSECALC_H
#define TOWNHOUSECALC_H

#include <abstractcalc.h>

class townhouseCalc : public abstractCalc
{
public:
```

```
    virtual double getCost(estate* value);

};

#endif // TOWNHOUSECALC_H
```

townhousecalc.cpp:

```
#include "townhousecalc.h"

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

abstractCalc.h:

```
#ifndef ABSTRACTCALC_H
#define ABSTRACTCALC_H

#include <estate.h>
//рефакторинг с помощью двух классов
class abstractCalc
{
public:
    abstractCalc();
    virtual double getCost(estate* value) = 0;
    virtual ~abstractCalc() {}
};

#endif // ABSTRACTCALC_H

//стоим
// для каждого объекта свой класс дом коттедж и тд + свой метод геткост
```

abstractCalc.cpp:

```
#include "abstractcalc.h"

abstractCalc::abstractCalc()
{

}
```

apartmentFactory.h:

```
#ifndef APARTMENTFACTORY_H
#define APARTMENTFACTORY_H

#include <calcfactory.h>

class apartmentFactory : public calcFactory
{
public:
    abstractCalc* createCalc();
    ~apartmentFactory() {}
};

#endif // APARTMENTFACTORY_H
```

apartmentFactory.cpp:

```
#include "apartmentfactory.h"

abstractCalc* apartmentFactory::createCalc(){
    return new apartmentCalc;
}
```

calcFactory.h:

```
#ifndef CALCFACTORY_H
#define CALCFACTORY_H

#include <apartmentcalc.h>
#include <cottagecalc.h>
#include <luxuriouscalc.h>
#include <townhousecalc.h>

class calcFactory
{
public:
    calcFactory();
    virtual abstractCalc* createCalc() = 0;
    virtual ~calcFactory() {}

};

#endif // CALCFACTORY_H

//рефакторинг с помощью двух классов

//создание объектов нужных для вычисления + своя именная факторка
```

calcFactory.cpp:

```
#include "calcfactory.h"

calcFactory::calcFactory()
{

}
```

cottageFactory.h:

```
#ifndef COTTAGEFACTORY_H
#define COTTAGEFACTORY_H

#include <calcfactory.h>

class cottageFactory : public calcFactory
{
public:
    abstractCalc* createCalc();
    ~cottageFactory() {}
};

#endif // COTTAGEFACTORY_H
```

cottageFactory.cpp:

```
#include "cottagefactory.h"

abstractCalc* cottageFactory::createCalc(){
    return new cottageCalc;
}
```

luxuriousFactory.h:

```
#ifndef LUXURIOUSFACTORY_H
#define LUXURIOUSFACTORY_H

#include <calcfactory.h>

class luxuriousFactory : public calcFactory
{
public:
    abstractCalc* createCalc();
    ~luxuriousFactory() {}
};

#endif // LUXURIOUSFACTORY_H
```

luxuriousFactory.cpp:

```
#include "luxuriousfactory.h"

abstractCalc* luxuriousFactory::createCalc(){
    return new luxuriousCalc;
}
```

townhouseFactory.h:

```
#ifndef TOWNHOUSEFACTORY_H
#define TOWNHOUSEFACTORY_H

#include <calcfactory.h>

class townhouseFactory : public calcFactory
{
public:
    abstractCalc* createCalc();
    ~townhouseFactory() {}
};

#endif // TOWNHOUSEFACTORY_H
```

townhouseFactory.cpp:

```
#include "townhousefactory.h"

abstractCalc* townhouseFactory::createCalc(){
    return new townhouseCalc;
}
```

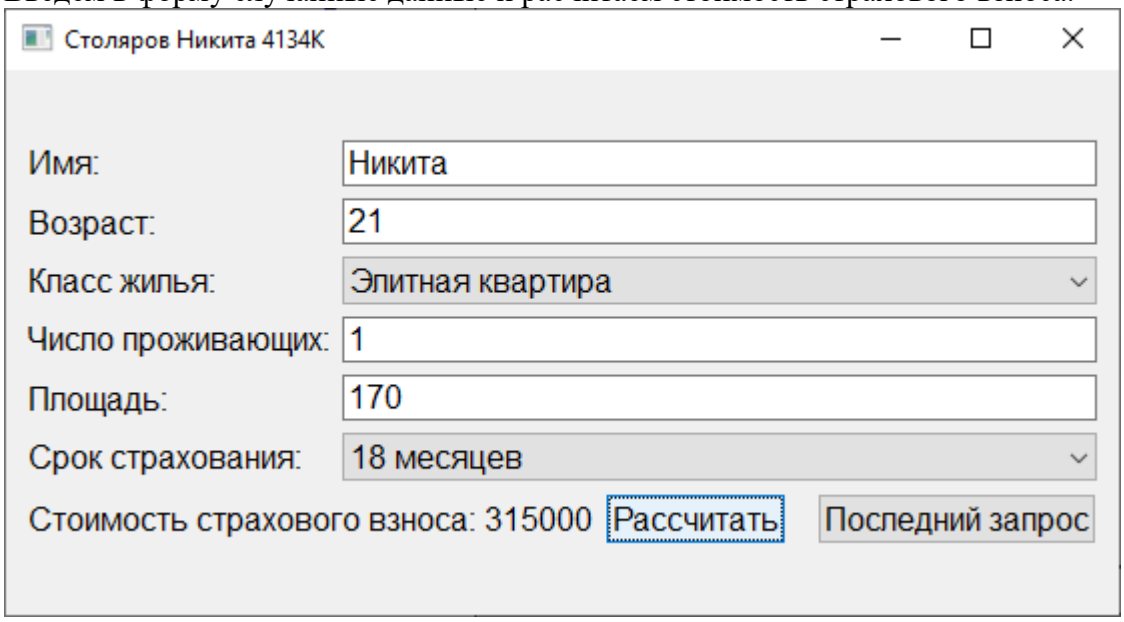
exception.h:

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <QException>
#include <QMessageBox>

class myException : public QException
{
public:
    myException(QString const &text = " ") noexcept : msg(text) {}
    myException(const myException &err) { this->msg = err.msg; }
    ~myException() override {}
```

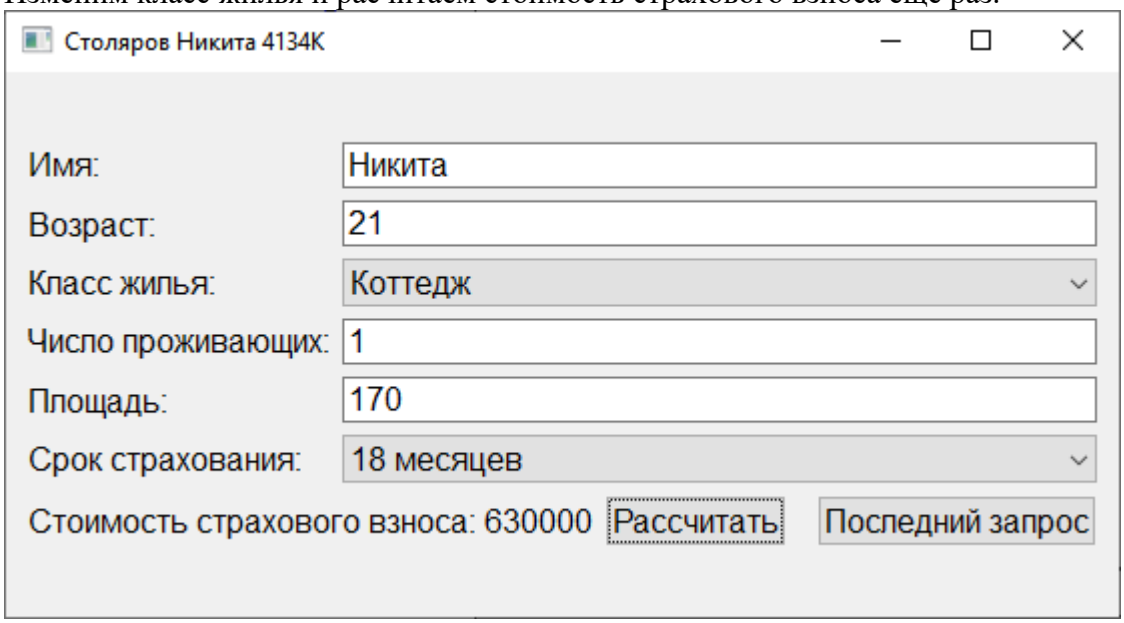
```
    void raise() const override { throw *this; }
    myException *clone() const override { return new myException(*this); }
    const char *what() const noexcept override { return this-
>msg.toStdString().c_str(); }
private:
    QString msg;
};

#endif // EXCEPTION_H
```

6.Пример выполнения программы:

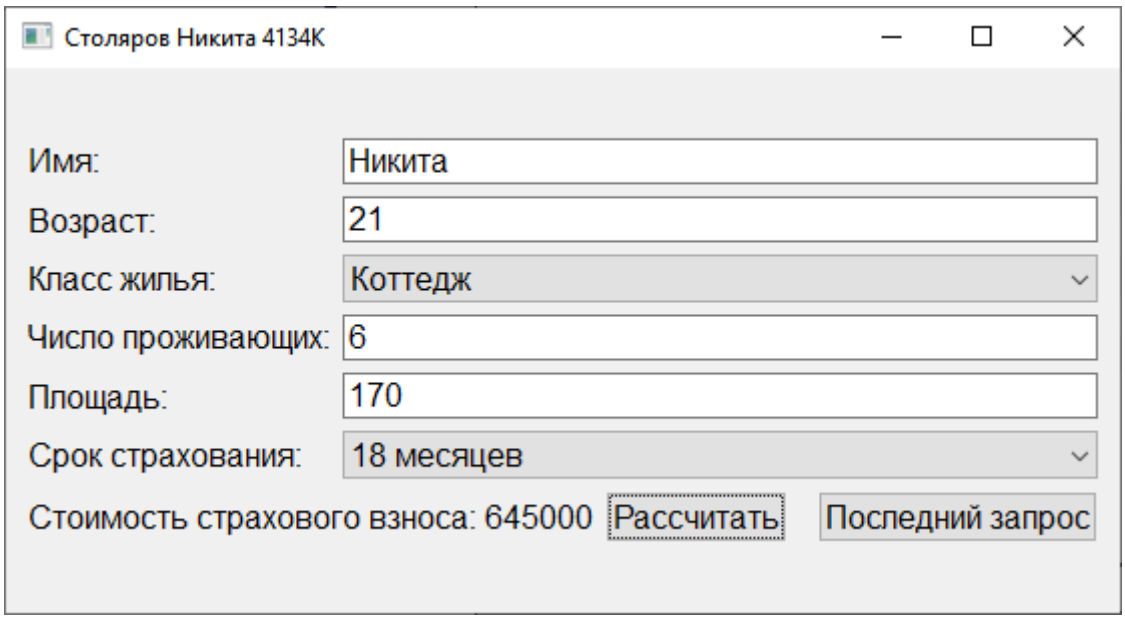
Введем в форму случайные данные и расчитаем стоимость страхового взноса.



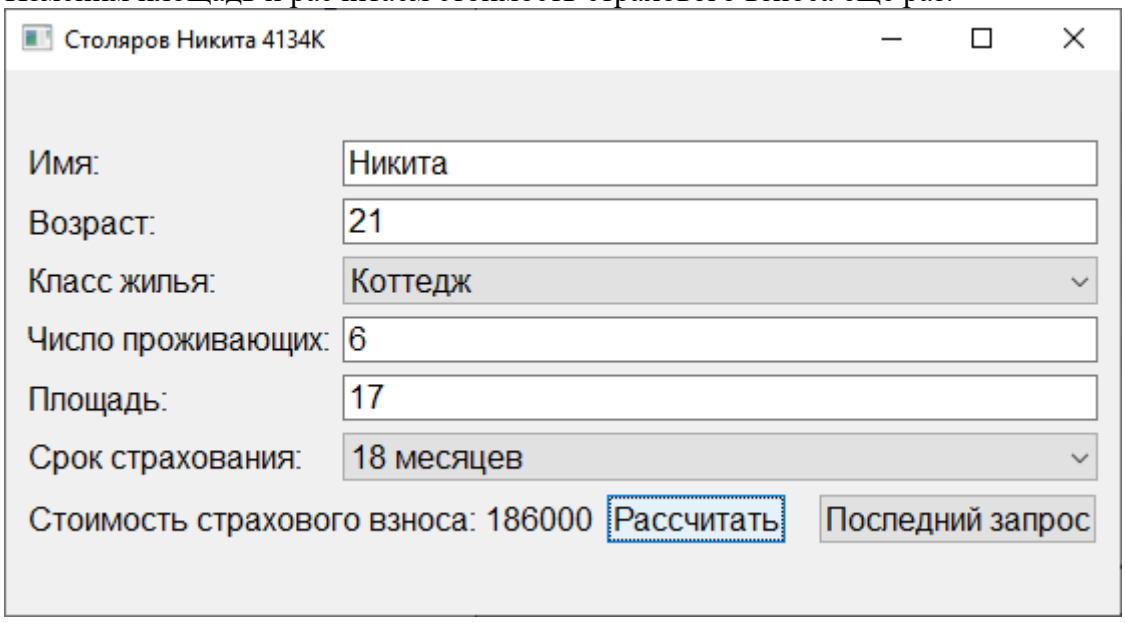
Изменим класс жилья и расчитаем стоимость страхового взноса еще раз.



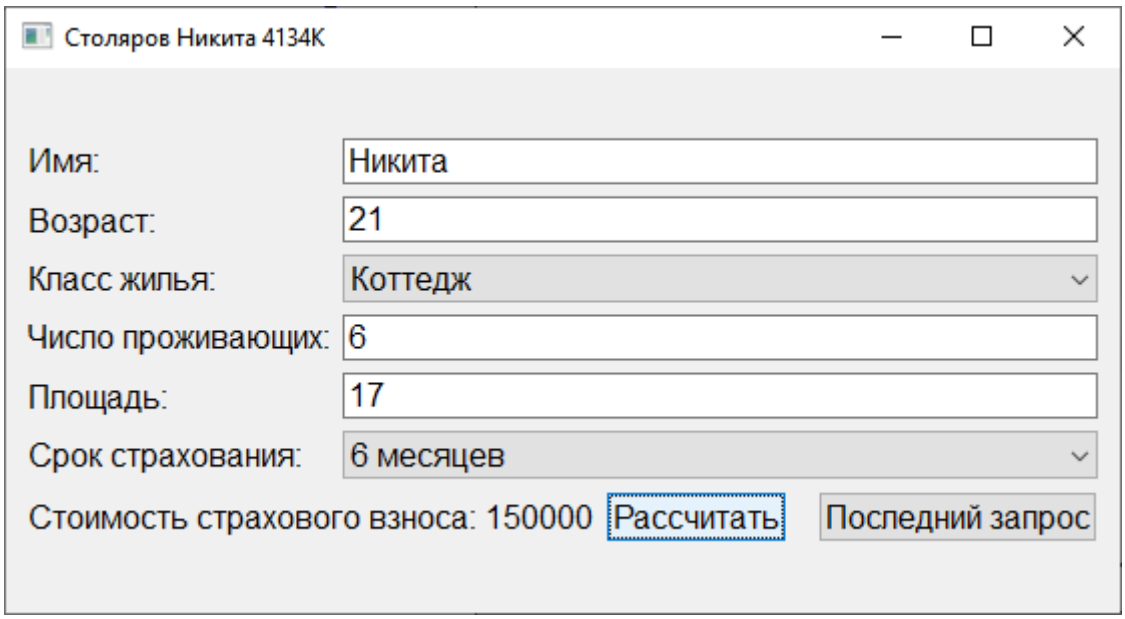
Изменим число проживающих и расчитаем стоимость страхового взноса еще раз.



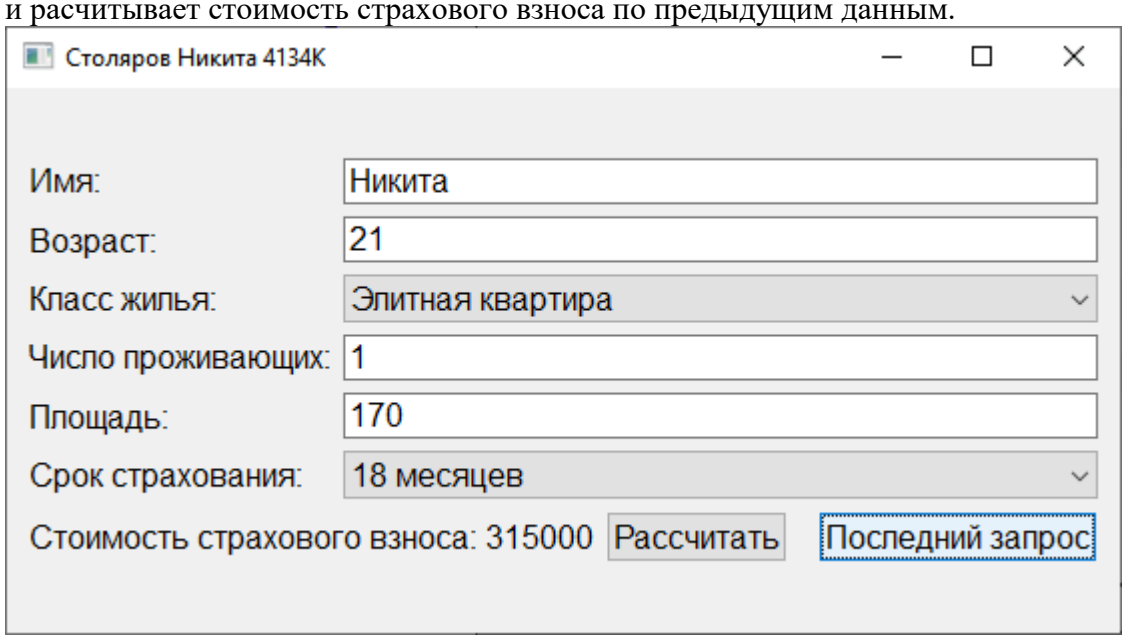
Изменим площадь и расчитаем стоимость страхового взноса еще раз.



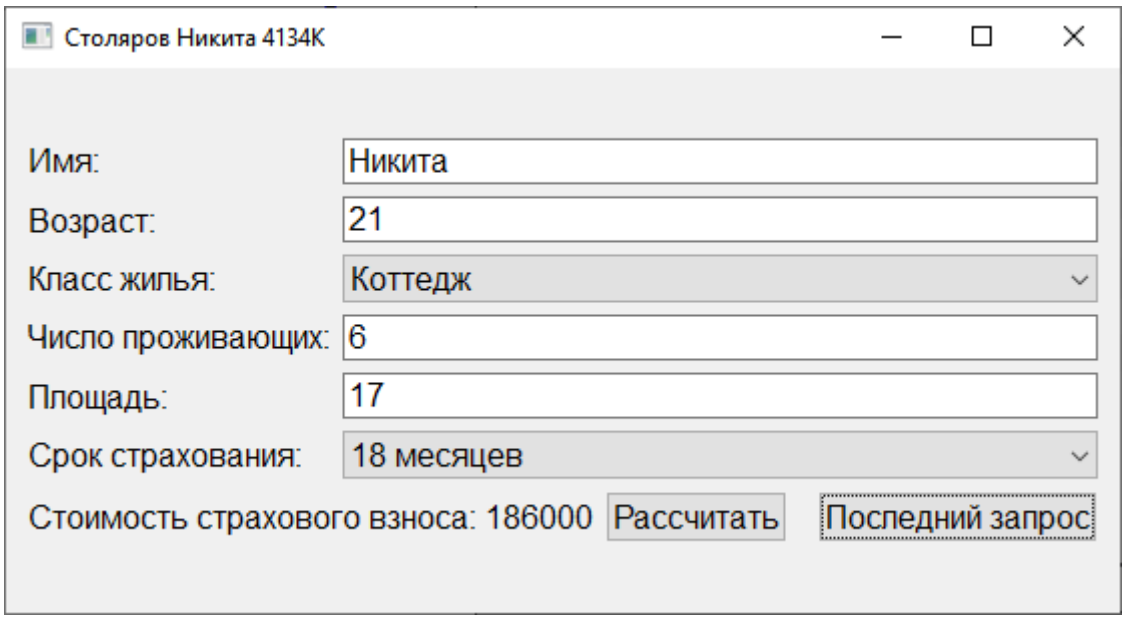
Изменим срок страхования и расчитаем стоимость страхового взноса еще раз.



Вернемся к предыдущему запросу. Форма заполняется данными из предыдущего запроса и расчитывает стоимость страхового взноса по предыдущим данным.



Вернемся к первому запросу. Кнопка блокируется потому что предыдущих запросов не имеется.



7.Анализ результатов и выводы:

В ходе этой лабораторной работы мы изучили принципы построения приложений с графическим интерфейсом, использую библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.