ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Ст.преподаватель		Шумова Е.О.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 6

Структурные и поведенческие шаблоны проектирования

по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4331		Кумпель А.Г
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1. **Цель работы**

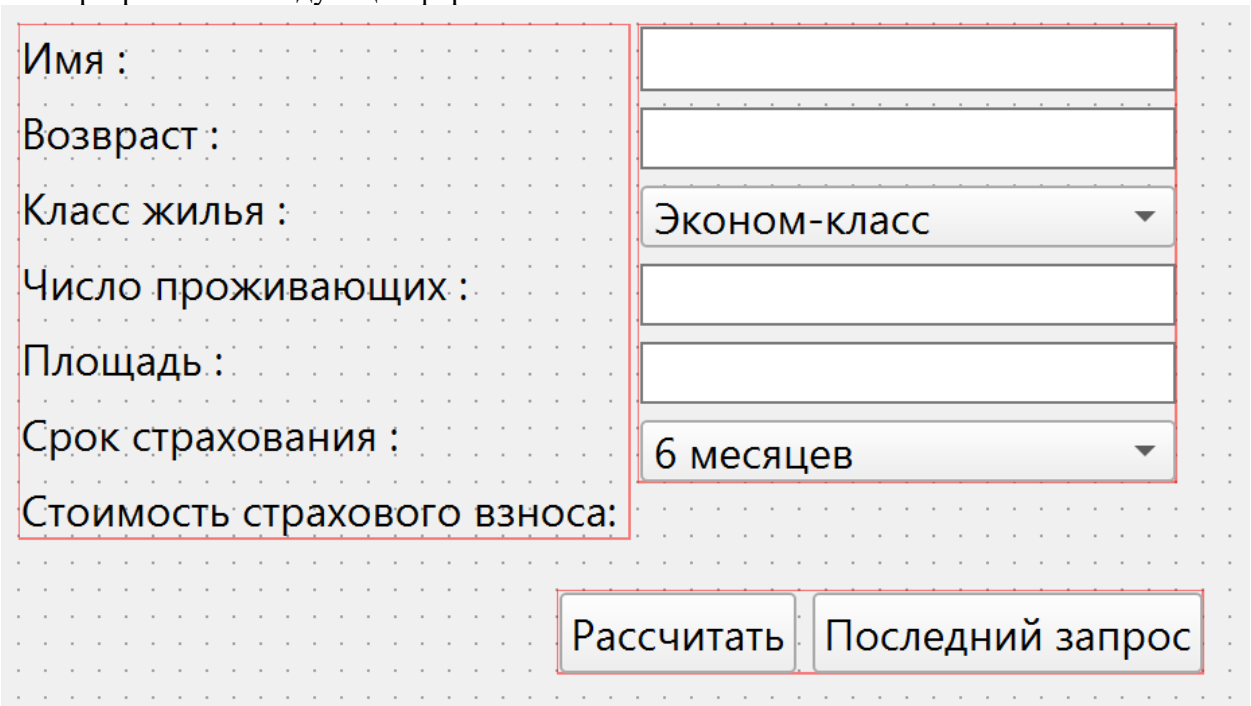
Изучить принципы построения приложений с графическим интерфейсом, использую библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

2. **Задание на лабораторную работу**

Выполнить все требования к лабораторной работе, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу. Подготовить отчёт в соответствии с шаблоном.

3. **Ход работы**

Была разработана следующая форма:



Пояснения к виджетам(сверху вниз).

1.lineedit owner для ввода имени человека который оплачивает страховку с соответствующей подсказкой слева.

2.lineedit age для ввода возвраста с подсказкой слева.

3.QcomboBox estateType для ввода типа жилья с подсказкой слева.

4.lineedit residents для ввода количества жильцов с подсказкой слева.

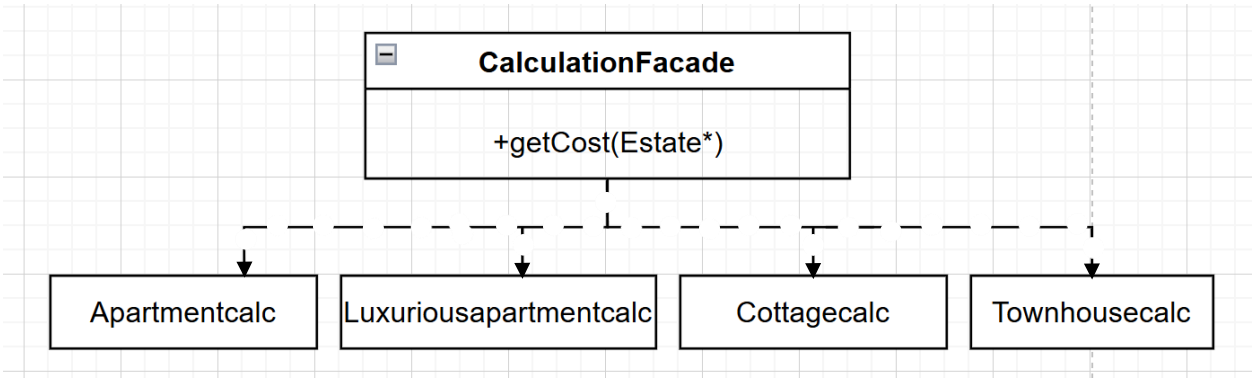
5.lineedit area для ввода площади пространства для страхования с подсказкой слева.

6.QcomboBox period для ввода периода страхования с подсказкой слева.

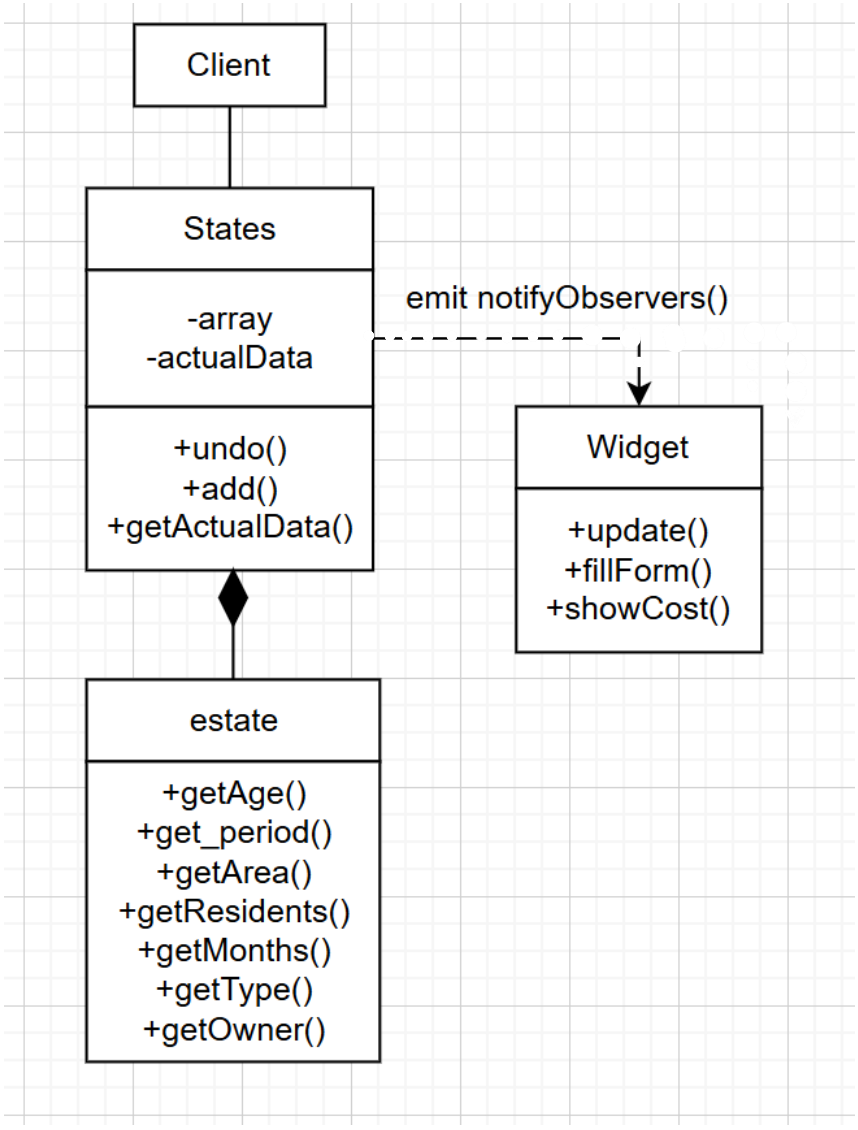
7.pushButton-ы btnCalc и btnUndo.Первая кнопка рассчитывает общую стоимость страхования за весь период, вторая – возвращает предыдущий запрос(предыдущего пользователя который запрашивал страхование).

Диаграммы классов для паттернов проектирования:

Вычисления(фасад):



Наблюдатель:



Листинги программ.

Файлы вычислений для конкретного типа зданий:

#include "estate.h"

```
class apartmentcalc {
public:
   static int getCost(Estate* estate) {
      return (estate->getArea() * 1000 + estate->getResidents() * 500)*0.01*estate->get_period();
   }
};

#include "estate.h"

class cottagecalc {
public:
   static int getCost(Estate* estate) {
      return (estate->getArea() * 3000 + estate->getResidents() * 1500)*0.01*estate-
>get_period();
   }
};
#include "estate.h"

class luxuriousapartmentcalc {
public:
   static int getCost(Estate* estate) {
      return (estate->getArea() * 2000 + estate->getResidents() * 1000)*0.01*estate-
>get_period();
   }
};
#include "estate.h"

class townhousecalc {
public:
   static int getCost(Estate* estate) {
      return (estate->getArea() * 4000 + estate->getResidents() * 2500)*0.01*estate-
>get_period();
   }
};
```

Header-файлы для остальных действий:

```
// Листинг файла calculationfacade.h
#ifndef CALCULATIONFACADE_H
#define CALCULATIONFACADE_H
#include <QObject>
#include "estate.h"
#include "apartmentcalc.h"
#include "luxuriousapartmentcalc.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
//Листинг файла estate.h
#ifndef ESTATE_H
#define ESTATE_H

#include <QObject>

class Estate : public QObject {
    Q_OBJECT
public:
    enum EstateType { ECONOM, LUXURIOUS, TOWN_HOUSE, COTTAGE };
    Estate(int age, int area, int residents, int months,EstateType type, const QString& owner,
QObject* parent);

    // Геттеры
    int get_period() const;
    int getAge() const;
    int getArea() const;
    int getResidents() const;
    QString getMonths() const;
    EstateType getType() const;
    QString getOwner() const;

private:
    int age;
    int area;
    int residents;
    int months;
    EstateType type;
    QString owner;
};

#endif // ESTATE_H

#ifndef STATES_H
#define STATES_H
// Листинг файла states.h
#include <QObject>
#include "estate.h"
class States : public QObject
{
    Q_OBJECT
public:
    explicit States(QObject* parent = nullptr);
    ~States();

    void undo();
    bool hasStates() const;
    Estate* getActualData() const;
    void add(Estate* value);

signals:
```

```
    void notifyObservers();

private:
    QList<Estate*> array;
    Estate* actualData;
};
#endif

//Листинг widget.h
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QComboBox>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include "estate.h"
#include "states.h"
#include "calculationfacade.h"

// Предварительное объявление класса Ui::Widget (генерируется автоматически Qt Designer)
namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

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

public slots:
    void update();

private slots:
    void onBtnCalcPressed();
    void onBtnUndoPressed();

private:
    Estate* processForm();       // Создание объекта Estate из данных формы
    void fillForm(Estate* value);      // Заполнение формы данными из Estate
    void showCost(int cost) const;     // Отображение рассчитанной стоимости
    Ui::Widget *ui;
    States history;                  // История запросов (паттерн Наблюдатель)
    CalculationFacade calculator;      // Фасад для расчетов
};
#endif
```

Cpp-файлы в том же порядке:

```
// Листинг файла calculationfacade.cpp
#include "calculationfacade.h"
CalculationFacade::CalculationFacade(QObject *parent) : QObject(parent)
{
}
int CalculationFacade::getCost(Estate *value)
{
   int cost;
   switch (value->getType()) {
   case Estate::EstateType::ECONOM:
      cost = apartmentcalc::getCost(value);
      break;
   case Estate::EstateType::LUXURIOUS:
      cost = luxuriousapartmentcalc::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;
}
#include "estate.h"
//Листинг estate.cpp
// Конструктор с инициализацией всех полей
Estate::Estate(int age, int area, int residents, int months,
            EstateType type, const QString& owner, QObject* parent):

QObject(parent),age(age),area(area),residents(residents),months(months+((months==1)*11)),typ
e(type),owner(owner) {
}

// Геттеры для полей
int Estate::getAge() const { return age; }
int Estate::getArea() const { return area; }
int Estate::getResidents() const { return residents; }
QString Estate::getMonths() const {
   if(months==6) return "6 месяцев";
   if(months==12) return "1 год";
   if(months==18) return "18 месяцев";
}
Estate::EstateType Estate::getType() const { return type; }
int Estate::get_period() const {return months;}
QString Estate::getOwner() const { return owner; }
//Листинг states.cpp
#include "states.h"
```

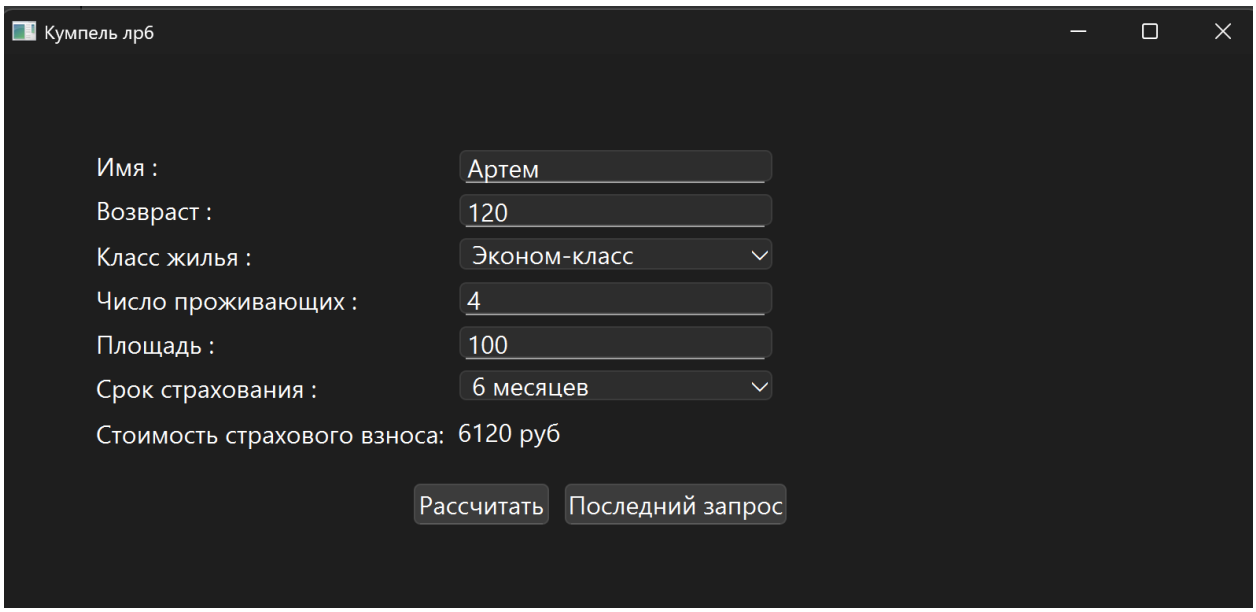
```
States::States(QObject *parent) : QObject(parent)
{
    actualData = nullptr;
}
States::~States() {
    qDeleteAll(array); // Удаляем все объекты
    array.clear();
    delete actualData; // Удаляем текущие данные
}
void States::add(Estate* value) {
    array.append(value);    // Добавляем в QList
    actualData = value;     // Обновляем текущие данные
    notifyObservers(); // Уведомляем наблюдателей
}

// Отмена последнего действия
void States::undo() {
    if (!array.isEmpty()) {
        actualData = array.takeAt(array.length()-2); // Берем последний элемент
        notifyObservers();
    } else {
        actualData = nullptr;
    }
}

// Проверка наличия элементов в коллекции
bool States::hasStates() const {
    return array.length()-1;
}

// Получение текущих данных
Estate* States::getActualData() const {
    return actualData;
}
//Листинг widget.cpp
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
    , history(this) // Инициализация history с указанием родителя
    , calculator(this) // Инициализация calculator с указанием родителя
{
    ui->setupUi(this); // Настройка пользовательского интерфейса

    // Начальное состояние кнопки Undo
    ui->btnUndo->setEnabled(false);

    // Подключение сигналов к слотам
    connect(ui->btnCalc, &QPushButton::clicked, this, &Widget::onBtnCalcPressed);
```

```
    connect(ui->btnUndo, &QPushButton::clicked, this, &Widget::onBtnUndoPressed);
    connect(&history, &States::notifyObservers, this, &Widget::update);
}

Widget::~Widget()
{
    delete ui; // Освобождение памяти, занятой интерфейсом
}

// Обновление интерфейса при изменении данных
void Widget::update()
{
    Estate* current = history.getActualData();
    if (current) {
        fillForm(current);
    }
    ui->btnUndo->setEnabled(history.hasStates());
}

// Обработка нажатия кнопки расчета
void Widget::onBtnCalcPressed()
{
    Estate* estate = processForm();
    if (estate->getAge()!=0 && estate->getArea()!=0 && estate->get_period()!=0 && estate-
>getOwner()!="" &&estate->getResidents()!=0) {
        int cost = calculator.getCost(estate);
        showCost(cost);
        history.add(estate);
        ui->btnUndo->setEnabled(true);
    }
    else ui->cost->setText("Проверьте введенность данных!");
}

// Обработка нажатия кнопки отмены
void Widget::onBtnUndoPressed() {
    if (history.hasStates()) {
        history.undo();
        Estate* estate = processForm();
        if (estate) {
            int cost = calculator.getCost(estate);
            showCost(cost);
        }
    }
}

// Создание объекта Estate из данных формы
Estate* Widget::processForm() {
    return new Estate(
        ui->age->text().toInt(),
        ui->area->text().toInt(),
        ui->residents->text().toInt(),
        ui->period->currentText().mid(0,2).toInt(),
```

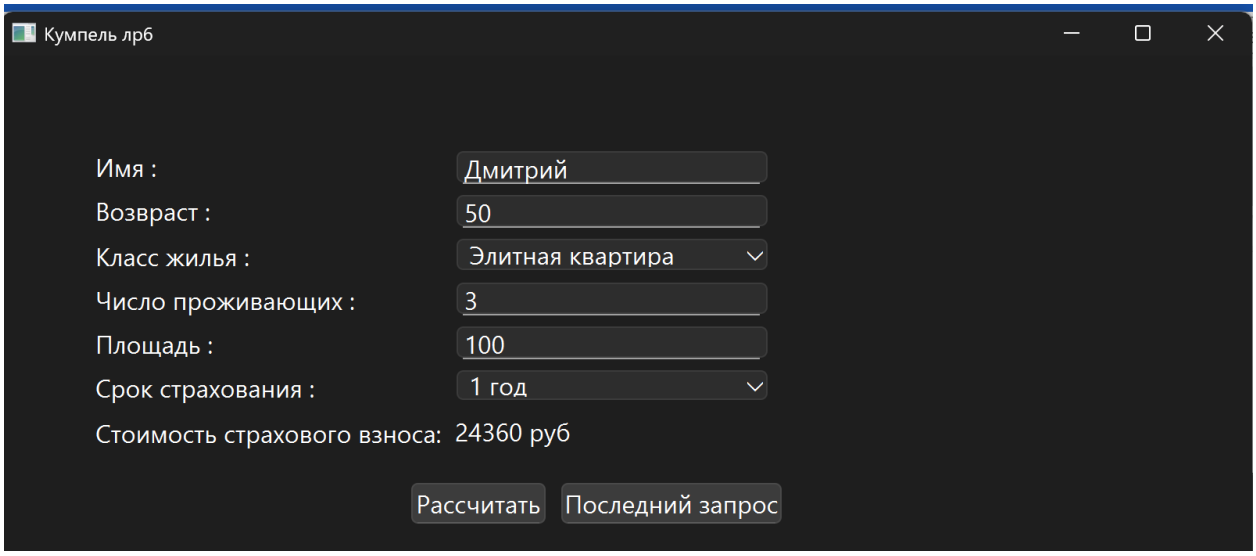
```
        static_cast<Estate::EstateType>(ui->estateType->currentIndex()),
        ui->owner->text(),
        this // Родительский объект
        );
}

// Заполнение формы данными из объекта Estate
void Widget::fillForm(Estate* value)
{
    if (!value) return;

    ui->age->setText(QString::number(value->getAge()));
    ui->area->setText(QString::number(value->getArea()));
    ui->residents->setText(QString::number(value->getResidents()));
    ui->period->setCurrentText(value->getMonths());
    ui->estateType->setCurrentIndex(static_cast<int>(value->getType()));
    ui->owner->setText(value->getOwner());

}

// Отображение рассчитанной стоимости
void Widget::showCost(int cost) const
{
    ui->cost->setText(QString::number(cost)+" руб");
}
//Листинг main.cpp
#include "widget.h"

#include <QApplication>

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

1 Примеры работы программы

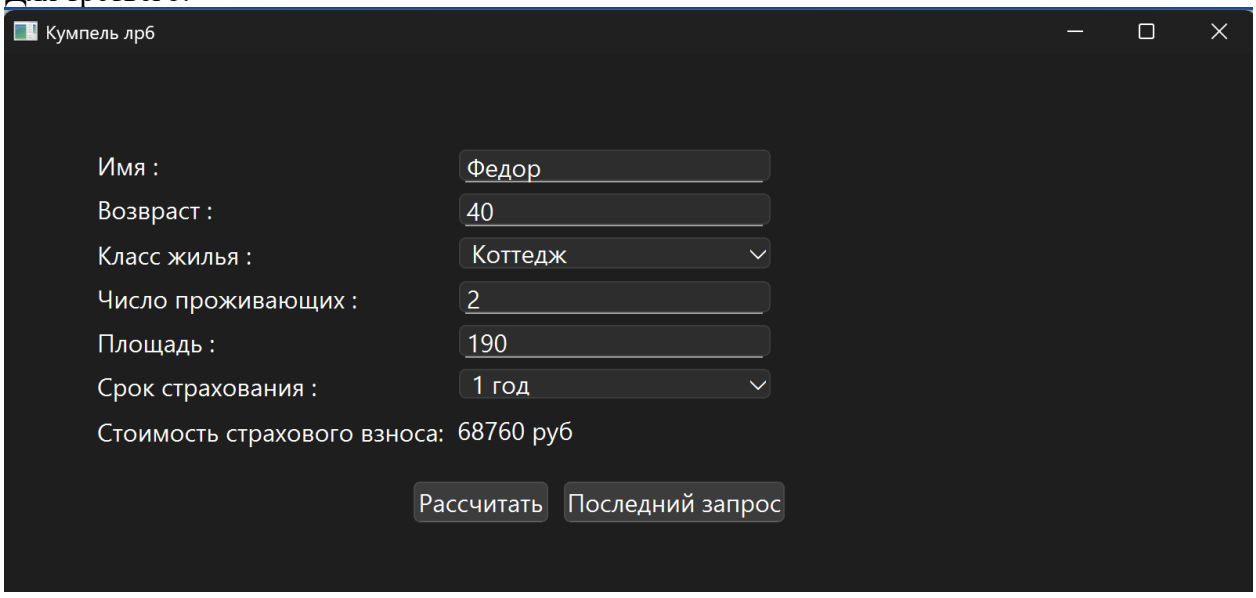
Введем данные для 1-го пользователя



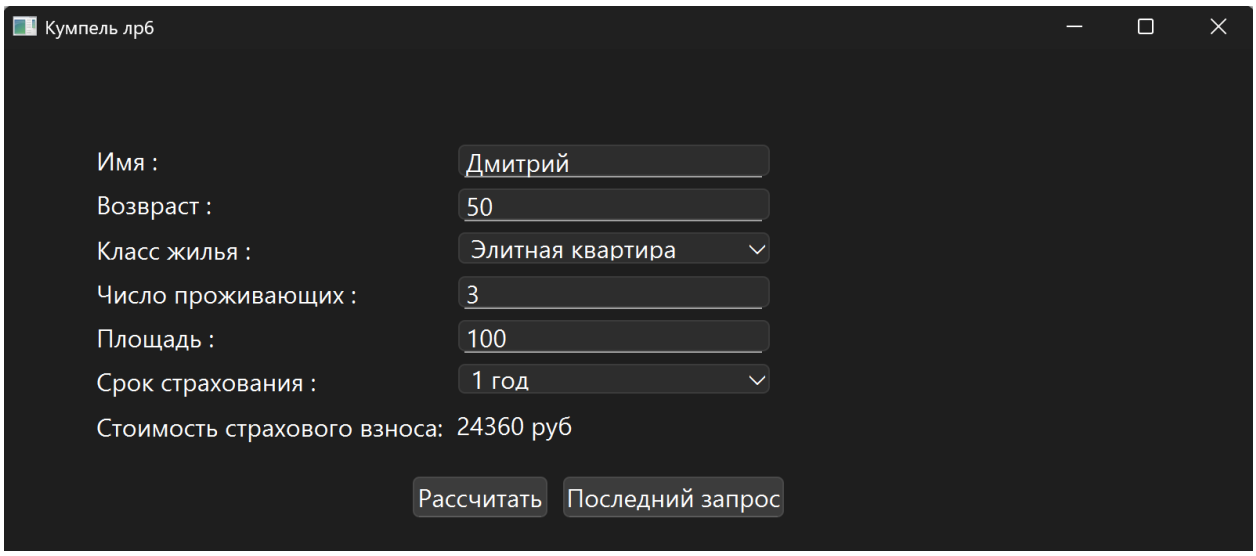
Теперь для второго:



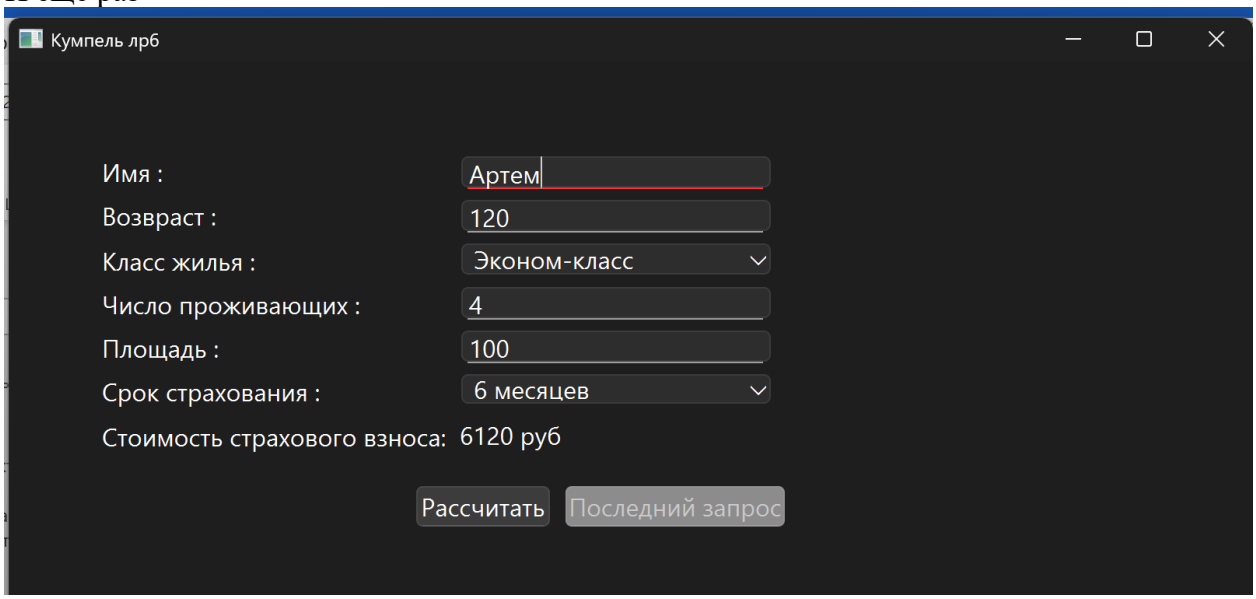
Для третьего:



Теперь попробуем получить последний запрос(ожидаем дмитрия)



И еще раз



Выводы:

В ходе выполнения лабораторной работы, я изучил принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Я научился выстраивать свои проекты “Фасадным” методом, который позволяет создавать более закрытые от пользователя проекты. Хотя в данной работе я не считаю использование этого метода необходимым, такие методы очень полезны при построении очень крупных проектов.