

1. Задание

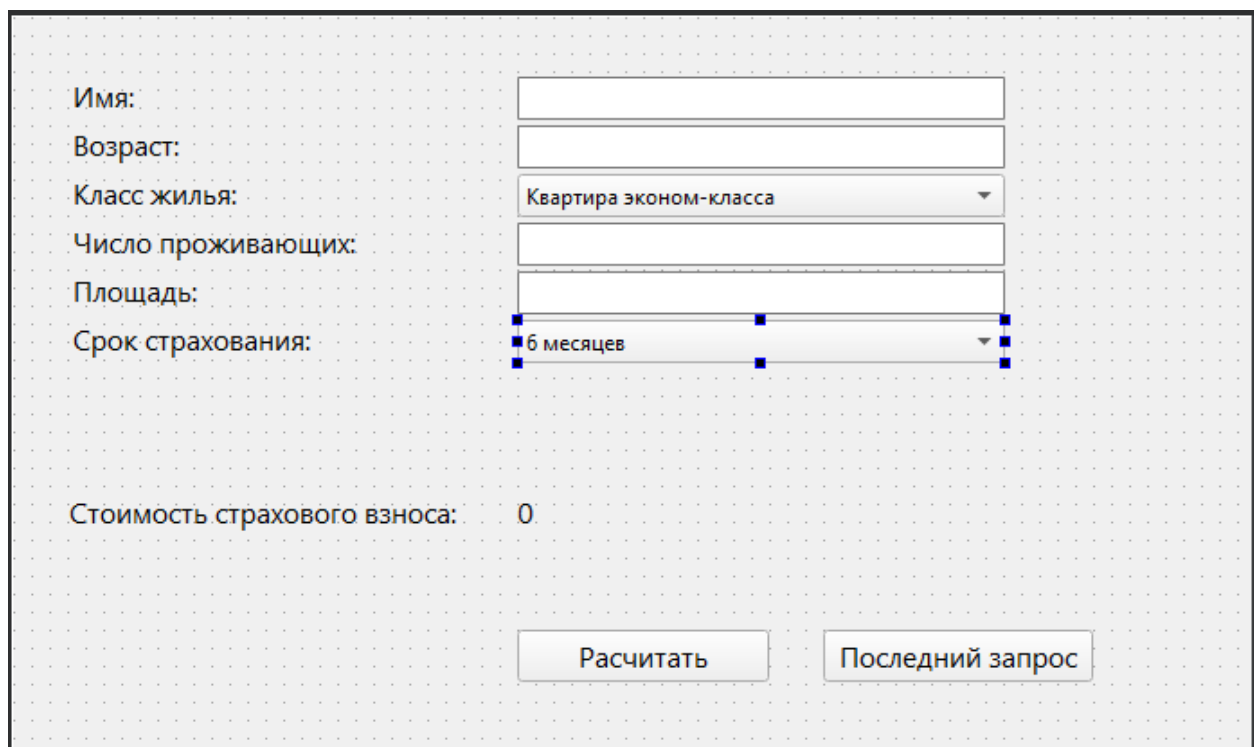
Задание 1. Дополните класс Estate конструктором, заполняющим все его поля и функциями чтения полей.

Задание 2. Реализуйте методы undo, hasStates, getActualData, add

Задание 3. Реализуйте собственные функции расчета стоимости в отдельном классе каждая и организуйте их вызов из класса-фасада. Класс-фасад может быть изменен, но передача данных для расчета должна производиться в виде объекта Estate, как показано в примере выше.

Задание 4. Реализуйте функции processForm, fillForm, showCost. Убедитесь в том, что приложение реализует функциональность, заявленную во введении.

2. Форма программы



Имя:

Возраст:

Класс жилья:

Число проживающих:

Площадь:

Срок страхования:

Стоимость страхового взноса: 0

Рисунок 1 – Форма

3. Описание формы

- QLabel – ageLabel – Название ввода возраста
- QLabel – areaLabel – Название ввода площади
- QLabel – houseLabel – Название выбора дома
- QLabel – insurPeriodLabel – Название выбора периода
- QLabel – insurPrice – Вывод цены
- QLabel – insurPriceLabel – Название вывода цены
- QLabel – LivingLabel – Название ввода проживающих

- QLabel – nameLabel – Название ввода возраста
- QLineEdit – age – Ввод возраста
- QLineEdit – area – Ввод площади
- QLineEdit – living – Ввод живущих
- QLineEdit – name – Ввод имени
- QComboBox - house – Выбор дома
- QComboBox - insurPeriod – Выбор периода
- QPushButton – calculaye – Произвести расчёт
- QComboBox - lastInput – Кнопка последнего расчёта

4. Диаграммы классов

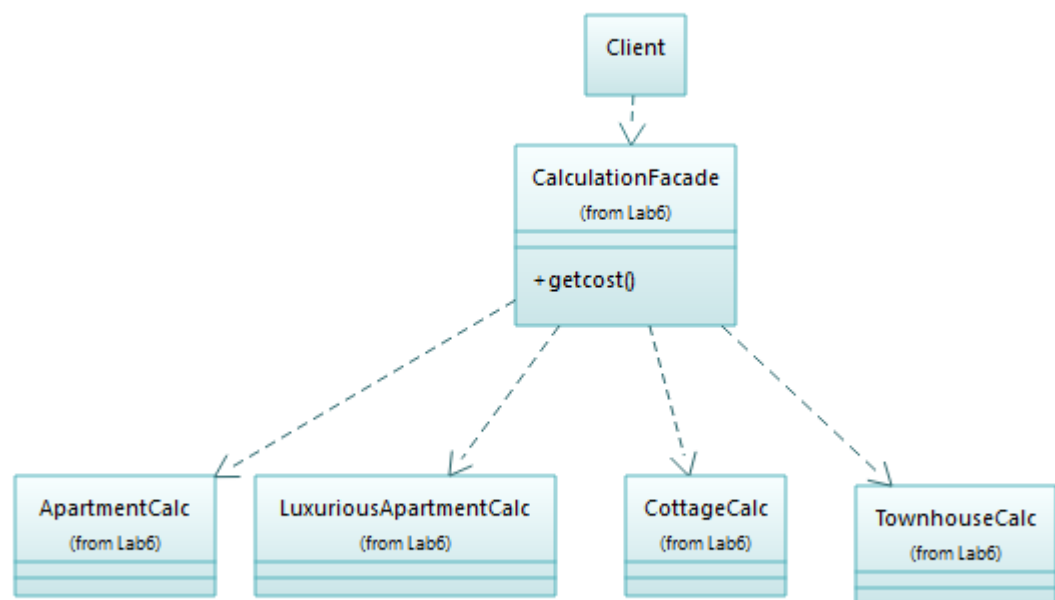


Рисунок 2 – Диаграмма паттерна Фасад

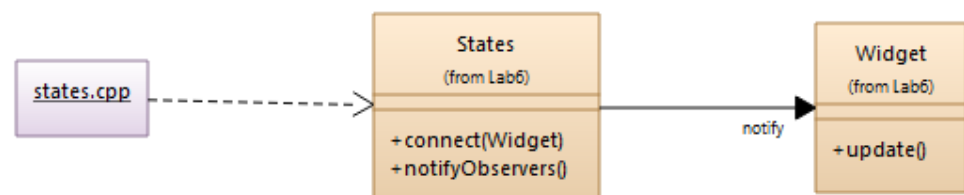


Рисунок 3 – Диаграмма паттерна Наблюдатель

5. Текст программы

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);
```

```
signals:
```

```
};
```

```
#endif // APARTMENTCALC_H
```

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
```

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);

signals:
};
```

```
#endif // COTTAGECALC_H
```

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(QObject *parent = nullptr, int = 0, int = 0, int = 0, int = 0, EstateType =  
EstateType::ECONOM, QString = "");
```

```
EstateType getType() const;  
int getAge() const;  
int getArea() const;  
int getResidents() const;  
int getMonths() const;  
QString getOwner() const;
```

private:

```
int age;  
int area;  
int residents;  
int months;  
EstateType type;  
QString owner;  
};
```

#endif // ESTATE_H

luxuriousapartment.h

```
#ifndef LUXURIOUSAPARTMENTCALC_H  
#define LUXURIOUSAPARTMENTCALC_H
```

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

```

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

signals:
};

#endif // LUXURIOUSAPARTMENTCALC_H

```

```

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
```

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);
```

```
signals:
};
```

```
#endif // TOWNHOUSECALC_H
```

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
```

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

```

#include <calculationfacade.h>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

public slots:
    void update();
    void updateCost();

private slots:
    void btnCalcPressed();
    void btnUndoPressed();

private:
    Estate *processForm();
    void fillForm(Estate *value);
    void showCost(Estate *value);

private:
    Ui::Widget *ui;
    States info;
};

#endif // WIDGET_H

```


apartmentcalc.cpp

```
#include "apartmentcalc.h"
```

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

```
// Расчёт стоимости для обычной квартиры
```

```
int ApartmentCalc::getCost(Estate *value)
{
    float k = 1;
    int price_per_p = 75;
    int square_k = 60;
    int price_per_6m = 100;
    return k * (price_per_p * value->getResidents() + square_k * value->getArea() +
price_per_6m * value->getMonths());
}
```

calculationfacade.cpp

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

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

```
// Фасад для расчёта стоимости
```

```
int CalculationFacade::getCost(Estate *value)
{
    int cost;
    switch (value->getType()) {
    case Estate::EstateType::ECONOM: // Обычная квартира
        cost = ApartmentCalc::getCost(value);
```

```

        break;
    case Estate::EstateType::LUXURIOUS: // Люкс квартира
        cost = LuxuriousApartmentCalc::getCost(value);
        break;
    case Estate::EstateType::TOWN_HOUSE: // ТаунХаус
        cost = TownhouseCalc::getCost(value);
        break;
    case Estate::EstateType::COTTAGE: // Коттедж
        cost = CottageCalc::getCost(value);
        break;
    default:
        cost = -1;
        break;
    }
    return cost;
}

```

cottagecalc.cpp

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

```

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

```

```
// Расчёт стоимости для коттеджа
```

```

int CottageCalc::getCost(Estate *value)
{
    float k = 1.65;
    int price_per_p = 165;
    int square_k = 185;
    int price_per_6m = 400;
    return k * (price_per_p * value->getResidents() + square_k * value->getArea() +
price_per_6m * value->getMonths());
}

```

estate.cpp

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

```
// Конструктор класса
```

```
Estate::Estate(QObject *parent, int age, int area, int residents, int months, EstateType type,  
QString owner)
```

```
    : QObject{parent}
```

```
{
```

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

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

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

```
    this->months = months;
```

```
    this->type = type;
```

```
    this->owner = owner;
```

```
}
```

```
// Функции получения атрибутов
```

```
Estate::EstateType Estate::getType() const
```

```
{
```

```
    return type;
```

```
}
```

```
int Estate::getAge() const
```

```
{
```

```
    return age;
```

```
}
```

```
int Estate::getArea() const
```

```
{
```

```
    return area;
```

```
}
```

```
int Estate::getResidents() const
```

```
{  
    return residents;  
}
```

```
int Estate::getMonths() const  
{  
    return months;  
}
```

```
QString Estate::getOwner() const  
{  
    return owner;  
}
```

```
luxuriousapartment.cpp  
#include "luxuriousapartmentcalc.h"
```

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

```
// Расчёт стоимости для люкс квартиры
```

```
int LuxuriousApartmentCalc::getCost(Estate *value)  
{  
    float k = 1.2;  
    int price_per_p = 100;  
    int square_k = 95;  
    int price_per_6m = 200;  
    return k * (price_per_p * value->getResidents() + square_k * value->getArea() +  
price_per_6m * value->getMonths());  
}
```

```
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"
```

```
States::States(QObject *parent) : QObject(parent)
```

```
{
```

```
    actualData = nullptr;
```

```
}
```

```
// Деструктор(удаление данных в списке)
```

```
States::~States()
```

```
{
```

```
    // delete: actualData
```

```
    if (actualData)
```

```
    {
```

```
        delete actualData;
```

```
        actualData = nullptr;
```

```
    }
```

```
    // delete and clear: array
```

```
    qDeleteAll(array);
```

```
    array.clear();
```

```
}
```

```
// Метод получения информации, есть ли в списке что-либо
```

```
bool States::hasStates()
```

```

{
    return array.size() != 0;
}

// Метод добавления в список нового элемента
void States::add(Estate *value)
{
    array.append(value);
    actualData = value;
}

// Метод получения последнего элемента
Estate* States::getActualData()
{
    return actualData;
}

// Метод удаления последнего элемента
void States::undo()
{
    if (not(hasStates()))
    {
        actualData = nullptr;
    }
    else
    {
        actualData = array.takeLast();
    }
    emit notifyObservers();
}

```

townhousecalc.cpp

```
#include "townhousecalc.h"
```

```

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

// Расчёт стоимости для таун хауса
int TownhouseCalc::getCost(Estate *value)
{
    float k = 1.4;
    int price_per_p = 135;
    int square_k = 115;
    int price_per_6m = 300;
    return k * (price_per_p * value->getResidents() + square_k * value->getArea() +
price_per_6m * value->getMonths());
}

```

widget.cpp

```

#include "widget.h"
#include "ui_widget.h"
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget),
    info(this)
{
    ui->setupUi(this);
    ui->lastInput->setEnabled(false);

    // регистрация слушателя
    connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));

    // регистрация остальных слотов и сигналов
    connect(ui->calculate, SIGNAL(pressed()), this, SLOT(btnCalcPressed()));
    connect(ui->lastInput, SIGNAL(pressed()), this, SLOT(btnUndoPressed()));

    connect(ui->name, SIGNAL(textChanged(QString)), this, SLOT(updateCost()));

```

```

connect(ui->age, SIGNAL(textChanged(QString)), this, SLOT(updateCost()));
connect(ui->area, SIGNAL(textChanged(QString)), this, SLOT(updateCost()));
connect(ui->living, SIGNAL(textChanged(QString)), this, SLOT(updateCost()));

connect(ui->house, SIGNAL(currentIndexChanged(int)), this, SLOT(updateCost()));
connect(ui->insurPeriod, SIGNAL(currentIndexChanged(int)), this, SLOT(updateCost()));
}

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

// Функция при получении сообщения
void Widget::update()
{
    // Получение последнего элемента в списке и заполнение формы
    auto value = info.getActualData();
    if(value != nullptr){
        fillForm(value);
    }
    // Включение-выключение кнопки на основе заполненности списка
    ui->lastInput->setEnabled(info.hasStates());

    value = nullptr;
}

// Нажатие кнопки расчёта
void Widget::btnCalcPressed()
{
    // Заполнение формы
    auto value = processForm();
    showCost(value);
    info.add(value);
    // Включение кнопки последнего запроса

```



```

    ui->lastInput->setEnabled(true);

    value = nullptr;
}

// Нажатие кнопки последнего запроса
void Widget::btnUndoPressed()
{
    info.undo();
}

// Заполнение данными объект Estate
Estate *Widget::processForm()
{
    int houseIndex = ui->house->currentIndex();
    Estate::EstateType type = Estate::EstateType(houseIndex);

    return new Estate(nullptr, ui->age->text().toInt(), ui->area->text().toInt(), ui->living->
    text().toInt(), (ui->insurPeriod->currentIndex()+1)*6, type, ui->name->text());
}

// Заполнение формы
void Widget::fillForm(Estate *value)
{
    ui->name->setText(value->getOwner());
    ui->age->setText(QString::number(value->getAge()));
    ui->living->setText(QString::number(value->getResidents()));
    ui->area->setText(QString::number(value->getArea()));
    ui->insurPeriod->setCurrentIndex(value->getMonths()/6 - 1);
    ui->house->setCurrentIndex((int)value->getType());
}

// Показ стоимости
void Widget::showCost(Estate *value)

```

```

{
    ui->insurPrice->setText( QString::number( CalculationFacade::getCost(value) ) );
}

// Обновление цены при изменении данных
void Widget::updateCost()
{
    ui->insurPrice->setText("0");
}

```

6. Работа программы

ВорошиловДаниил4333К

Имя:	<input type="text" value="Даниил"/>
Возраст:	<input type="text" value="19"/>
Класс жилья:	<input type="text" value="Элитная квартира"/>
Число проживающих:	<input type="text" value="3"/>
Площадь:	<input type="text" value="76"/>
Срок страхования:	<input type="text" value="6 месяцев"/>

Стоимость страхового взноса: 10464

Рисунок 4 – Расчёт данных

ВорошиловДаниил4333К

Имя: Даниил

Возраст: 19

Класс жилья: Элитная квартира

Число проживающих: 3

Площадь: 76

Срок страхования: 1 год

Стоимость страхового взноса: 11904

Расчитать Последний запрос

Рисунок 5 – Расчёт на основе других данных

ВорошиловДаниил4333К

Имя: Даниил

Возраст: 19

Класс жилья: Коттедж

Число проживающих: 3

Площадь: 76

Срок страхования: 1 год

Стоимость страхового взноса: 0

Расчитать Последний запрос

Рисунок 6 – Изменение данных

ВорошиловДаниил4333К

Имя: Даниил

Возраст: 19

Класс жилья: Элитная квартира

Число проживающих: 3

Площадь: 76

Срок страхования: 6 месяцев

Стоимость страхового взноса: 10464

Расчитать Последний запрос

Рисунок 7 – Возврат в самое начало с помощью кнопки Последний запрос

7. Выводы

В процессе лабораторной работы произошло ознакомление со структурными и поведенческими шаблонами проектирования и получены практические навыки их использования