ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | 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

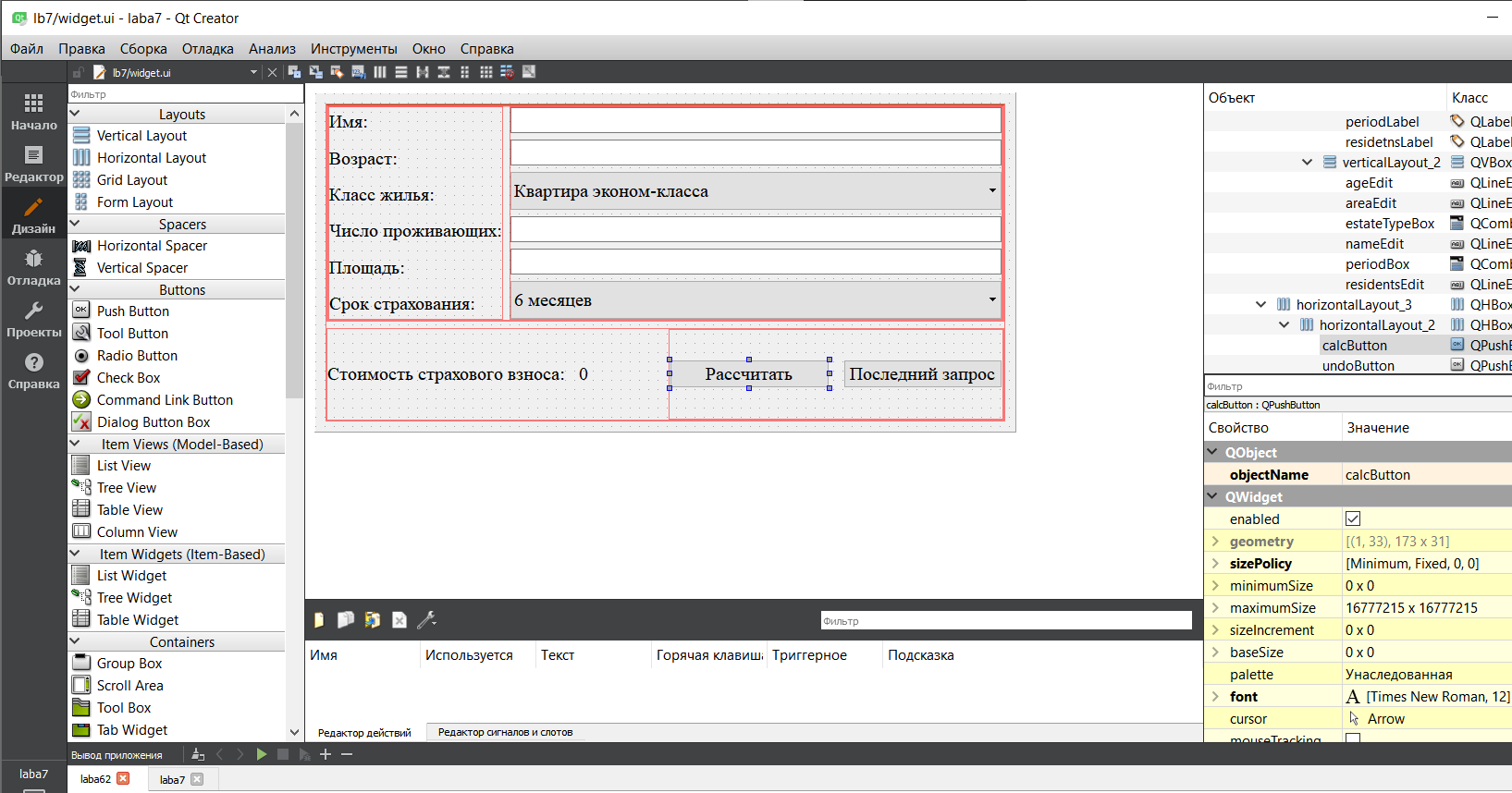
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

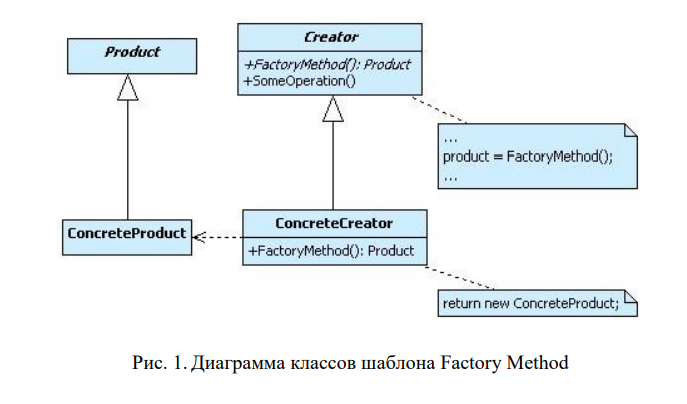
# Цель работы

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

# Вид исходной формы

****

**Диаграмма классов для паттерна проектирования**

****

# Листинг программы

abstractcalc.cpp

#include *"abstractcalc.h"*

abstractCalc::abstractCalc()

{

}

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*

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

apartmentcalc.cpp

#include *"apartmentcalc.h"*

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

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

}

apartmentcalc.h

#ifndef APARTMENTCALC\_H

#define APARTMENTCALC\_H

#include *<abstractcalc.h>*

**class** **apartmentCalc** : **public** abstractCalc

{

**public**:

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

};

#endif *// APARTMENTCALC\_H*

apartmentfactory.cpp

#include *"apartmentfactory.h"*

abstractCalc\* apartmentFactory::createCalc(){

**return** **new** apartmentCalc;

}

apartmentfactory.h

#ifndef APARTMENTFACTORY\_H

#define APARTMENTFACTORY\_H

#include *<calcfactory.h>*

**class** **apartmentFactory** : **public** calcFactory

{

**public**:

abstractCalc\* createCalc();

~apartmentFactory() {}

};

#endif *// APARTMENTFACTORY\_H*

calcfactory.cpp

#include *"calcfactory.h"*

calcFactory::calcFactory()

{

}

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*

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**:{

**break**;

}

}

**return** house->getCost(value);

}

*// обстрактный дом + нужный тип данных для вычислений, для подключения нужной фабрики*

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*

*// ститические объекты для избежания утечек памяти*

cottagecalc.cpp

#include *"cottagecalc.h"*

double cottageCalc::getCost(estate \*value){

**return** (value->getAge() + value->getArea() + value->getMonths() + value->getResidents()) \* 3000;

}

cottagecalc.h

#ifndef COTTAGECALC\_H

#define COTTAGECALC\_H

#include *<abstractcalc.h>*

**class** **cottageCalc** : **public** abstractCalc

{

**public**:

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

};

#endif *// COTTAGECALC\_H*

cottagefactory.cpp

#include *"cottagefactory.h"*

abstractCalc\* cottageFactory::createCalc(){

**return** **new** cottageCalc;

}

cottagefactory.h

#ifndef COTTAGEFACTORY\_H

#define COTTAGEFACTORY\_H

#include *<calcfactory.h>*

**class** **cottageFactory** : **public** calcFactory

{

**public**:

abstractCalc\* createCalc();

~cottageFactory() {}

};

#endif *// COTTAGEFACTORY\_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** myException("Заполните все поля формы.");

**this**->age = age;

**this**->area = area;

**this**->residents = residents;

**this**->months = months.split(" ")[0].toInt();

**this**->owner = owner;

**this**->type = **static\_cast**<EstateType>(type);

}

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**;

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** **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*

luxuriouscalc.cpp

#include *"luxuriouscalc.h"*

double luxuriousCalc::getCost(estate \*value){

**return** (value->getAge() + value->getArea() + value->getMonths() + value->getResidents()) \* 1500;

}

luxuriouscalc.h

#ifndef LUXURIOUSCALC\_H

#define LUXURIOUSCALC\_H

#include *<abstractcalc.h>*

**class** **luxuriousCalc** : **public** abstractCalc

{

**public**:

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

};

#endif *// LUXURIOUSCALC\_H*

luxuriousfactory.cpp

#include *"luxuriousfactory.h"*

abstractCalc\* luxuriousFactory::createCalc(){

**return** **new** luxuriousCalc;

}

luxuriousfactory.h

#ifndef LUXURIOUSFACTORY\_H

#define LUXURIOUSFACTORY\_H

#include *<calcfactory.h>*

**class** **luxuriousFactory** : **public** calcFactory

{

**public**:

abstractCalc\* createCalc();

~luxuriousFactory() {}

};

#endif *// LUXURIOUSFACTORY\_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"*

states::states(QObject \*parent)

: QObject{parent}

{

actualData = **nullptr**;

}

states::~states(){

**if** (actualData){

**delete** actualData;

actualData = **nullptr**;

}

qDeleteAll(array);

array.clear();

}

bool states::hasStates(){

**return** !(array.isEmpty());

}

estate\* states::getActualData(){

**return** actualData;

}

void states::add(estate\* value){

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

{

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"*

double townhouseCalc::getCost(estate \*value){

**return** (value->getAge() + value->getArea() + value->getMonths() + value->getResidents()) \* 2500;

}

townhousecalc.h

#ifndef TOWNHOUSECALC\_H

#define TOWNHOUSECALC\_H

#include *<abstractcalc.h>*

**class** **townhouseCalc** : **public** abstractCalc

{

**public**:

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

};

#endif *// TOWNHOUSECALC\_H*

townhousefactory.cpp

#include *"townhousefactory.h"*

abstractCalc\* townhouseFactory::createCalc(){

**return** **new** townhouseCalc;

}

townhousefactory.h

#ifndef TOWNHOUSEFACTORY\_H

#define TOWNHOUSEFACTORY\_H

#include *<calcfactory.h>*

**class** **townhouseFactory** : **public** calcFactory

{

**public**:

abstractCalc\* createCalc();

~townhouseFactory() {}

};

#endif *// TOWNHOUSEFACTORY\_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("^([А-Я][а-я]+)**\\**s([А-Я][а-яА-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** myException &error){

QMessageBox msg;

msg.setWindowTitle("Ошибка!");

msg.setFixedSize(500,400);

msg.setText(error.what());

msg.exec();

**return**;

}

**if**(info.getSize() >= 2) ui->undoButton->setEnabled(true);

**if**(info.getSize() <= 1) ui->undoButton->setEnabled(false);

}

void Widget::undoPressed(){

**if** (info.getSize() > 1) info.undo();

**if**(info.getSize() <= 1) ui->undoButton->setEnabled(false);

**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;

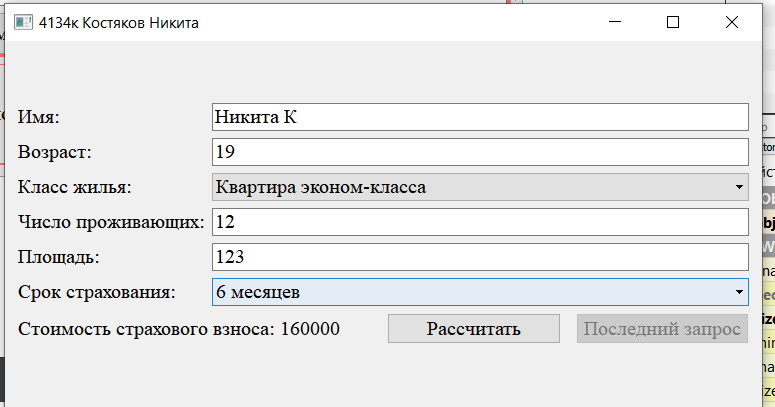
states info;

};

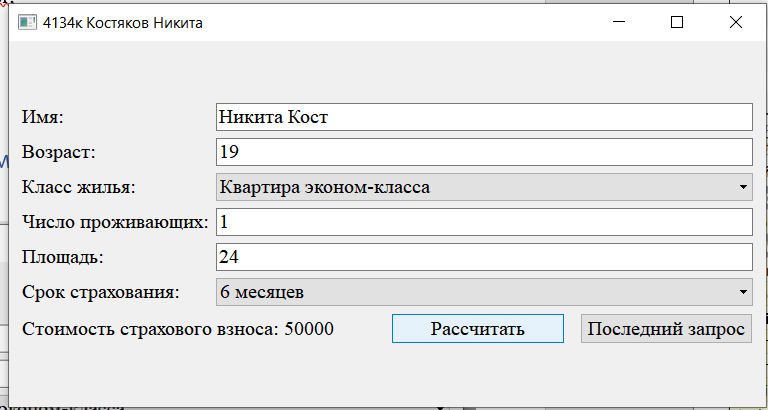
#endif *// WIDGET\_H*

# Результат работы программы

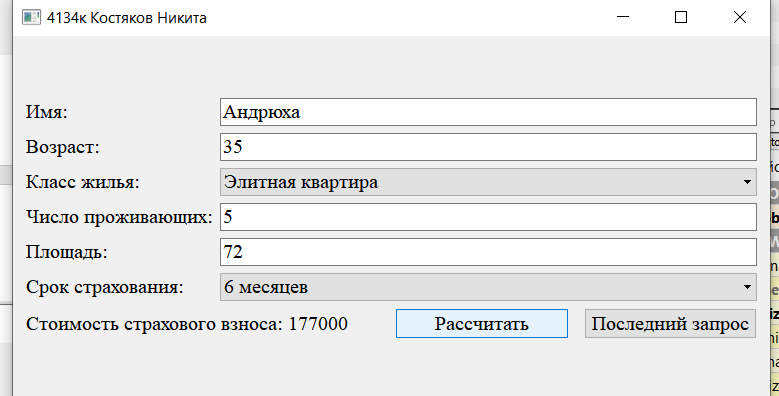
Расчет 1



Рассчет2

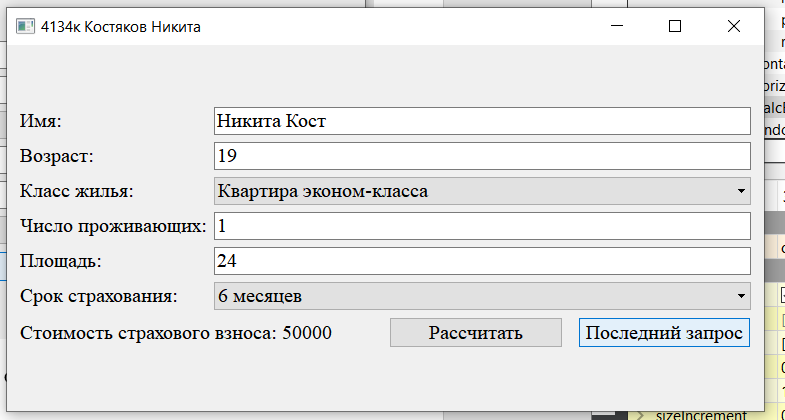


Расчет 3

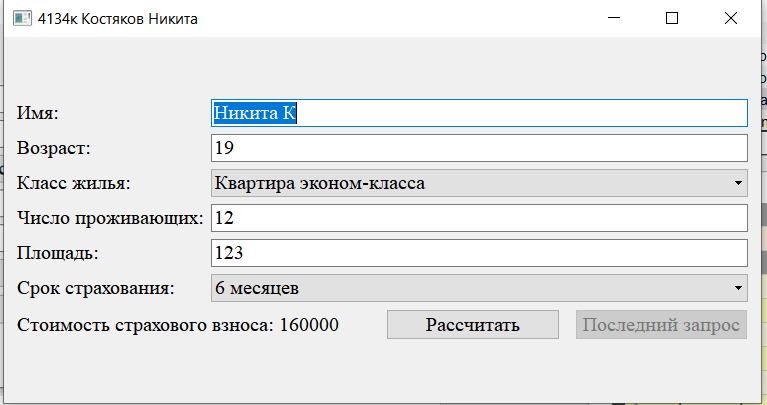


Работы кнопки Последнего запроса откатывает форму на предыдущий запрос

1 нажатие



Нажатие 2



# Выводы

Я изучил принципы построения приложений с графическим интерфейсом