МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЁТ ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | Кочин К.А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ № 7 |
| «Порождающие шаблоны проектирования» |
| по курсу: объектно-ориентированное программирование |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТЫ ГР. | 4136 |  |  |  | Н.С. Бобрович |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

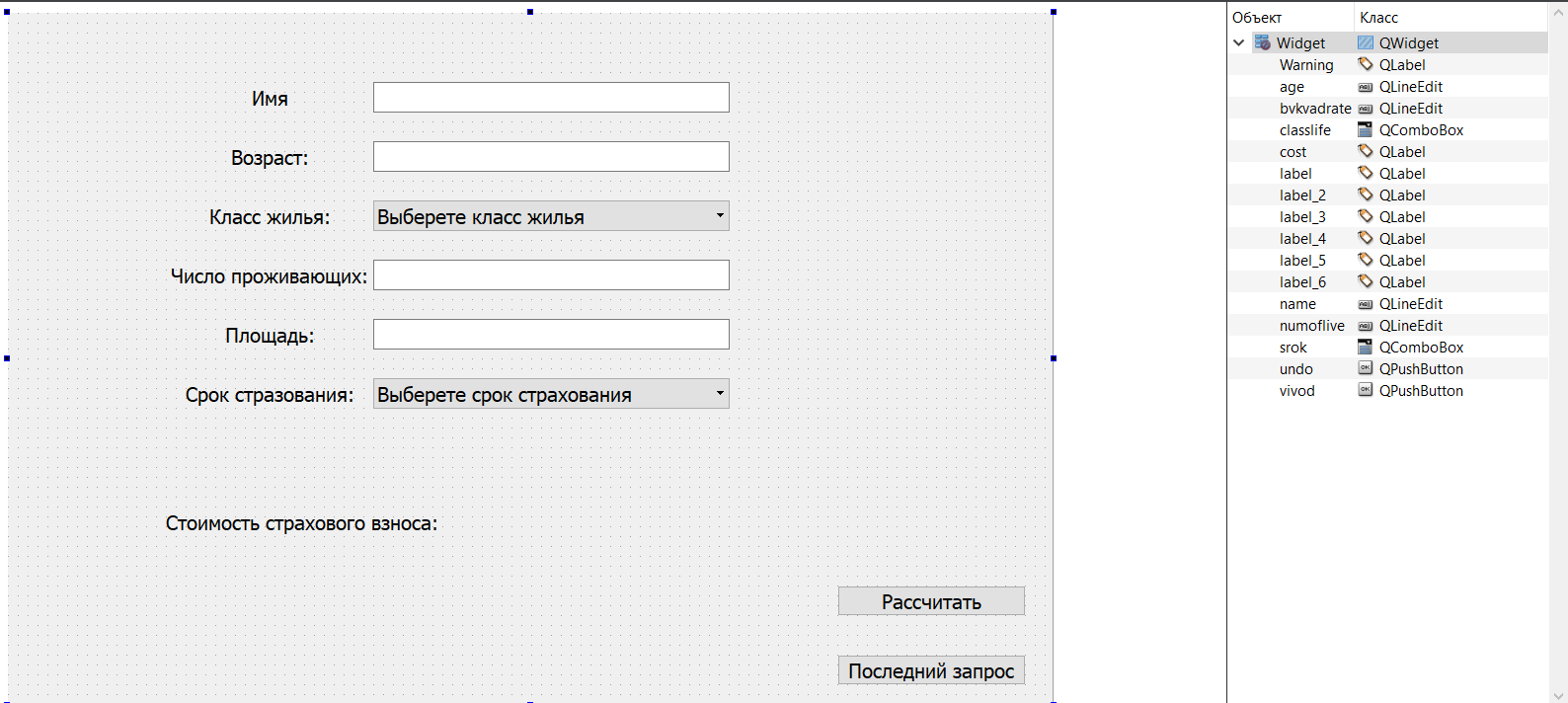
2023

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

**Задание:**

Необходимо изменить вызов статических функций getCost в CalculationFacade, на реализацию Factory Method. Для этого необходимо использовать базовые классы bstractCalc и CalcFactory реализующие интерфейсы паттерна Factory Method. В CalculationFacade, вместо статических функций будет вызываться фабрика необходимого для расчётов класса, создаваться объект класса, а у него вызов метода getCost.

**Вид формы в режиме дизайнера:**



**Описание используемых виджетов:**

Warning (QLabel) – поле для вывода ошибок и предупреждений

age (QLineEdit) – виджет, представляющий собой форму для ввода возраста

bvkvadrate (QLineEdit) – виджет, представляющий собой форму для ввода площади

classlife (QComboBox) – виджет, представляющий собой форму для выбора класса жилья

cost (QLabel) – виджет, представляющий собой поле для вывода стоимости страхового взноса

label (QLabel) - виджет, представляющий собой поле для вывода надписи “Имя”

label\_2 (QLabel) - виджет, представляющий собой поле для вывода надписи “Возраст:”

label\_3 (QLabel) - виджет, представляющий собой поле для вывода надписи “Класс жилья:”

label\_4 (QLabel) – виджет, представляющий собой поле для вывода надписи “Число проживающих:”

label\_5 (QLabel) - виджет, представляющий собой поле для вывода надписи “Площадь:”

label\_6 (QLabel) - виджет, представляющий собой поле для вывода надписи “Срок страхования:”

name (QLineEdit) – виджет, представляющий собой форму для ввода имени

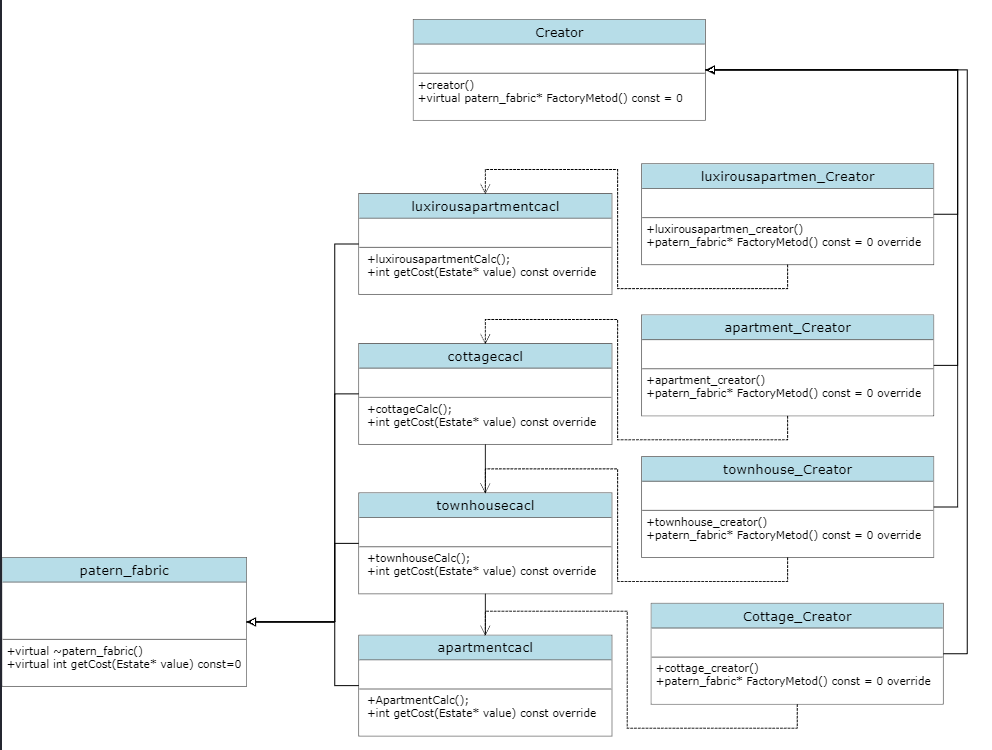
numoflife (QLineEdit) – виджет, представляющий собой форму для ввода количества жильцов

srok (QComboBox) – виджет, представляющий собой форму для выбора срока страхования

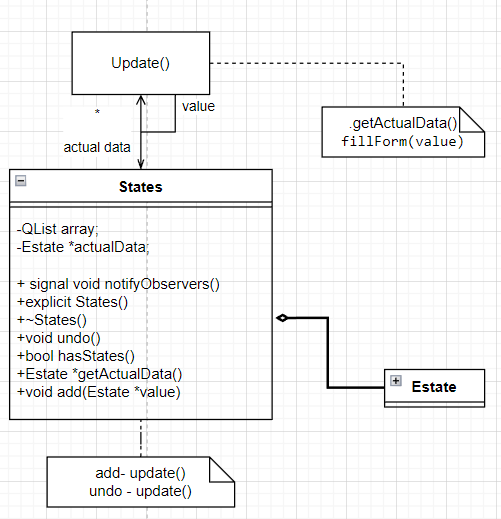
undo (QPushButton) – виджет, представляющий собой кнопку, которая заменяет все поля на предыдущие (предыдущий расчёт)

vivod (QPushButton) – виджет, представляющий собой кнопку, которая выводит стоимость страхового взноса и сохраняет данные

**Диаграммы классов:**

Для шаблона Fabric Method

Для шаблона Observer



**Программа:**

Apartment\_creator.h

#ifndef APARTMENT\_CREATOR\_H

#define APARTMENT\_CREATOR\_H

#include <creator.h>

#include <apartmentcalc.h>

class apartment\_creator : public creator

{

public:

apartment\_creator();

patern\_fabric\* *FactoryMetod*() const override;

};

#endif // APARTMENT\_CREATOR\_H

Apartment\_creator.cpp

#include "apartment\_creator.h"

apartment\_creator::apartment\_creator()

{

}

patern\_fabric\* apartment\_creator::*FactoryMetod*() const

{

return new ApartmentCalc;

}

Apartmentcalc.h

#ifndef APARTMENTCALC\_H

#define APARTMENTCALC\_H

#include <estate.h>

#include <patern\_fabric.h>

class ApartmentCalc : public patern\_fabric

{

public:

ApartmentCalc();

int *getCost*(Estate\* value) const override;

};

#endif // APARTMENTCALC\_H

Apartmentcalc.cpp

#include "apartmentcalc.h"

ApartmentCalc::ApartmentCalc()

{

}

int ApartmentCalc::*getCost*(Estate\* value) const//рассчёт стоимости при типе жилья Эконом-класс

{

int cost;

double square = value->getArea();

if (square < 50)

{

cost = 25000;

}

else

if (square < 100)

{

cost = 35000;

}

else

{

cost = 55000;

}

int age=value->getAge();

if (age>50)

cost+=2000\*value->getResidents();

else

cost+=3500\*value->getResidents();

int month= value->getMonths();

cost+=1600\*month;

return cost;

}

CalculationFacadec.h

#ifndef CALCULATIONFACADE\_H

#define CALCULATIONFACADE\_H

#include <QObject>

#include <estate.h>

#include <apartmentcalc.h>

#include <luxuriousapartmentcalc.h>

#include <townhousecalc.h>

#include <cottagecalc.h>

#include <creator.h>

#include <apartment\_creator.h>

#include <cottage\_creator.h>

#include <luxuriousapartment\_creator.h>

#include <townhouse\_creator.h>

class CalculationFacade : public QObject

{

Q\_OBJECT

public:

explicit CalculationFacade(QObject \*parent = nullptr);

static int getCost(Estate \*value);

};

#endif // CALCULATIONFACADE\_H

CalculationFacadec.cpp

#include "calculationfacade.h"

CalculationFacade::CalculationFacade(QObject \*parent) : QObject(parent)//конструктор класса

{

}

int CalculationFacade::getCost(Estate \*value)//рассчёт стоимости для переданного объекта класса Estate

{

int cost;

switch (value->getType())

{

case Estate::NOTHING://если нет типа жилья

cost=-1;

break;

case Estate::ECONOM://если тип жилья - эконом класс

{

creator\* econ = new apartment\_creator;

patern\_fabric\* econom = econ->*FactoryMetod*();

cost=econom->*getCost*(value);

break;

}

case Estate::LUXURIOUS://если тип жилья - Элитная квартира

{

creator\* econ = new cottage\_creator;

patern\_fabric\* econom = econ->*FactoryMetod*();

cost=econom->*getCost*(value);

}

break;

case Estate::TOWN\_HOUSE://если тип жилья таун-хаус

{

creator\* econ = new luxuriousapartment\_creator;

patern\_fabric\* econom = econ->*FactoryMetod*();

cost=econom->*getCost*(value);

break;

}

case Estate::COTTAGE://если тип жилья - коттедж

{

creator\* econ = new townhouse\_creator;

patern\_fabric\* econom = econ->*FactoryMetod*();

cost=econom->*getCost*(value);

break;

}

default:

cost=-1;

break;

}

return cost;

}

Cottage\_creator.h

#ifndef COTTAGE\_CREATOR\_H

#define COTTAGE\_CREATOR\_H

#include <creator.h>

#include <cottagecalc.h>

class cottage\_creator : public creator

{

public:

cottage\_creator();

patern\_fabric\* *FactoryMetod*() const override;

};

#endif // COTTAGE\_CREATOR\_H

Cottage\_creator.cpp

#include "cottage\_creator.h"

cottage\_creator::cottage\_creator()

{

}

patern\_fabric\* cottage\_creator::*FactoryMetod*() const

{

return new CottageCalc;

}

Cottagecalc.h

#ifndef COTTAGECALC\_H

#define COTTAGECALC\_H

#include <estate.h>

#include <patern\_fabric.h>

class CottageCalc : public patern\_fabric

{

public:

CottageCalc();

int *getCost*(Estate\* value) const override;

};

#endif // COTTAGECALC\_H

Cottagecalc.cpp

#include "cottagecalc.h"

CottageCalc::CottageCalc()

{

}

int CottageCalc::*getCost*(Estate\* value) const//рассчёт стоимости для класса жилья Коттедж

{

int cost;

double square = value->getArea();

if (square < 75)

{

cost = 120000;

}

else

if (square < 100)

{

cost = 155000;

}

else

{

cost = 215000;

}

int age=value->getAge();

if (age>50)

cost+=13500\*value->getResidents();

else

cost+=26000\*value->getResidents();

int month= value->getMonths();

cost+=10000\*month;

return cost;

}

Estate.h

#ifndef ESTATE\_H

#define ESTATE\_H

#include <QObject>

#include <QLabel>

class Estate : public QObject

{

Q\_OBJECT

public:

enum EstateType {

NOTHING,

ECONOM,

LUXURIOUS,

TOWN\_HOUSE,

COTTAGE

};

explicit Estate();

void GetInfo(QLabel\* textEdit, int cost);

void getall(int esttype, int forage, int forarea, int forresidents, int formonth, QString forowner);

Estate(int esttype, int forage, int forarea, int forresidents, int formonth, QString forowner, QObject \*parent);

EstateType getType();

int getAge();

int getArea();

int getResidents();

int getMonths();

QString getOwner();

private:

int age=-1;

int area=-1;

int residents=-1;

int months=-1;

EstateType type=EstateType::NOTHING;

QString owner="";

};

#endif // ESTATE\_H

Estate.cpp

#include "estate.h"

#include <QLabel>

Estate::Estate()//конструктор

{

age=-1;

area=-1;

residents=-1;

months=-1;

type=NOTHING;

owner="";

}

void Estate::getall(int esttype, int forage, int forarea, int forresidents, int formonth, QString forowner)//метод для заполнения всех полей переданными данными

{

switch (esttype) {

case 0:

type=EstateType::NOTHING;

break;

case 1:

type=EstateType::ECONOM;

break;

case 2:

type=EstateType::LUXURIOUS;

break;

case 3:

type=EstateType::TOWN\_HOUSE;

break;

case 4:

type=EstateType::COTTAGE;

break;

default:

type=EstateType::NOTHING;

break;

}

age=forage;

area=forarea;

residents=forresidents;

switch (formonth) {

case 0:

months=-1;

break;

case 1:

months=6;

break;

case 2:

months=12;

break;

case 3:

months=18;

break;

default:

months=-1;

break;

}

owner=forowner;

}

void Estate::GetInfo(QLabel \*textEdit, int cost)//функция для вывод стоимости взноса в форму, переданную в данный метод

{

textEdit->clear();

textEdit->setText("Стоимость страхового взноса: "+QString::number(cost));

}

Estate::Estate(int esttype, int forage, int forarea, int forresidents, int formonth, QString forowner, QObject \*parent) : QObject(parent)//конструктор, заполняющий все поля переданной информацией

{

switch (esttype) {

case 0:

type=NOTHING;

break;

case 1:

type=ECONOM;

break;

case 2:

type=LUXURIOUS;

break;

case 3:

type=TOWN\_HOUSE;

break;

case 4:

type=COTTAGE;

default:

type=NOTHING;

break;

}

age=forage;

area=forarea;

residents=forresidents;

switch (formonth) {

case 0:

months=-1;

break;

case 1:

months=6;

break;

case 2:

months=12;

break;

case 3:

months=18;

break;

default:

months=-1;

break;

}

owner=forowner;

}

Estate::EstateType Estate::getType()//получение типа жилья

{

return type;

}

int Estate::getAge()//возврат возраста

{

return age;

}

int Estate::getArea()//возврат площади

{

return area;

}

int Estate::getResidents()//возврат количества жильцов

{

return residents;

}

int Estate::getMonths()//возврат количества месяцев страхования

{

return months;

}

QString Estate::getOwner()//возврат имени

{

return owner;

}

Creator.h

#ifndef CREATOR\_H

#define CREATOR\_H

#include <patern\_fabric.h>

class creator

{

public:

creator();

virtual patern\_fabric\* *FactoryMetod*() const = 0;

};

#endif // CREATOR\_H

Creator.cpp

#include "creator.h"

creator::creator()

{

}

Luxuriousapartment\_creator.h

#ifndef LUXURIOUSAPARTMENT\_CREATOR\_H

#define LUXURIOUSAPARTMENT\_CREATOR\_H

#include <creator.h>

#include <luxuriousapartmentcalc.h>

class luxuriousapartment\_creator : public creator

{

public:

luxuriousapartment\_creator();

patern\_fabric\* *FactoryMetod*() const override;

};

#endif // LUXURIOUSAPARTMENT\_CREATOR\_H

Luxuriousapartment\_creator.cpp

#include "luxuriousapartment\_creator.h"

luxuriousapartment\_creator::luxuriousapartment\_creator()

{

}

patern\_fabric\* luxuriousapartment\_creator::*FactoryMetod*() const

{

return new LuxuriousApartmentCalc;

}

Luxuriousapartmentcalc.h

#ifndef LUXURIOUSAPARTMENTCALC\_H

#define LUXURIOUSAPARTMENTCALC\_H

#include <estate.h>

#include <patern\_fabric.h>

class LuxuriousApartmentCalc : public patern\_fabric

{

public:

LuxuriousApartmentCalc();

int *getCost*(Estate\* value) const override;

};

#endif // LUXURIOUSAPARTMENTCALC\_H

Luxuriousapartmentcalc.cpp

#include "luxuriousapartmentcalc.h"

LuxuriousApartmentCalc::LuxuriousApartmentCalc()//конструктор

{

}

int LuxuriousApartmentCalc::*getCost*(Estate\* value) const//рассчёт стоимости для типа жилья элитная квартира

{

int cost;

double square = value->getArea();

if (square < 50)

{

cost = 50000;

}

else

if (square < 100)

{

cost = 75000;

}

else

{

cost = 125000;

}

int age=value->getAge();

if (age>50)

cost+=7500\*value->getResidents();

else

cost+=15000\*value->getResidents();

int month= value->getMonths();

cost+=2200\*month;

return cost;

}

Patern\_fabric.h

#ifndef PATERN\_FABRIC\_H

#define PATERN\_FABRIC\_H

#include<estate.h>

class patern\_fabric

{

public:

virtual ~*patern\_fabric*();

virtual int *getCost*(Estate\* value) const=0;

};

#endif // PATERN\_FABRIC\_H

Patern\_fabric.cpp

#include "patern\_fabric.h"

patern\_fabric::~*patern\_fabric*()

{

}

States.h

#ifndef STATES\_H

#define STATES\_H

#include <QObject>

#include <QList>

#include <estate.h>

class States : public QObject

{

Q\_OBJECT

public:

explicit States();

~*States*();

void undo();

bool hasStates();

Estate \*getActualData();

void add(Estate \*value);

signals:

void notifyObservers();

private:

QList<Estate \*> array;

Estate \*actualData;

};

#endif // STATES\_H

States.cpp

#include "states.h"

States::States()//конструктор. текущие актуальные данные пусты

{

actualData = nullptr;

}

States::~*States*()//деструктор

{

// удаляем actualData

if (actualData)

{

delete actualData;

actualData = nullptr;

}

// удаляем и очищаем array

array.clear();

}

void States::undo()//возврат к предыдущему сохранённому состоянию формы

{

if (array.isEmpty())//если список пуст

{

actualData = nullptr;

}

else

{

actualData = array.last();//последние данные - последний элемент списка

array.removeLast();//данный элемент больше не находиться в списке

emit notifyObservers();//сигнал об изменении списка

}

}

bool States::hasStates()//проверка на наличие элементов в списке

{

return !array.isEmpty();

}

Estate \*States::getActualData()//получение актуальных данных

{

return actualData;

}

void States::add(Estate \*value)//добавление данных

{

// добавляем элемент в коллекцию

array.append(value);

// устанавливаем actualData в значение последнего элемента

actualData = value;

emit notifyObservers();//сигнал об изменении списка

}

Townhouse\_creator.h

#ifndef TOWNHOUSE\_CREATOR\_H

#define TOWNHOUSE\_CREATOR\_H

#include <creator.h>

#include <townhousecalc.h>

class townhouse\_creator : public creator

{

public:

townhouse\_creator();

patern\_fabric\* *FactoryMetod*()const override;

};

#endif // TOWNHOUSE\_CREATOR\_H

Townhousecreator.cpp

#include "townhouse\_creator.h"

townhouse\_creator::townhouse\_creator()

{

}

patern\_fabric\* townhouse\_creator::*FactoryMetod*() const

{

return new TownHouseCalc;

}

Townhousecalc.h

#ifndef TOWNHOUSECALC\_H

#define TOWNHOUSECALC\_H

#include <estate.h>

#include <patern\_fabric.h>

class TownHouseCalc : public patern\_fabric

{

public:

explicit TownHouseCalc();

int *getCost*(Estate \*value) const override;

};

#endif // TOWNHOUSECALC\_H

Townhousecalc.cpp

#include "townhousecalc.h"

TownHouseCalc::TownHouseCalc()

{

}

int TownHouseCalc::*getCost*(Estate \*value) const//метод для рассчёта стоимости при типе жилья таунхаус

{

int cost;

double square = value->getArea();//получение данных о площади

if (square < 50)//если помещение маленькое

{

cost = 1000000;

}

else

if (square < 100)//если помещение среднее

{

cost = 1500000;

}

else//если помещение большое

{

cost = 2000000;

}

int age=value->getAge();

if (age>50)//если жильцы старые

cost+=10000\*value->getResidents();

else//если жильцы молодые

cost+=20000\*value->getResidents();

int month= value->getMonths();

cost+=50000\*month;//стоимость равна количеству месяцев умноженному на стоимость месяца

return cost;

}

Wodget.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <estate.h>

#include <states.h>

#include <calculationfacade.h>

namespace Ui {

class Widget;

}

class Widget : public QWidget

{

Q\_OBJECT

public:

explicit Widget(QWidget \*parent = 0);

~*Widget*();

public slots:

void update();

private slots:

void agecheck();

void srokcheck();

void task();

void namecheck();

void ploscheck();

void lifecheck();

void numcheck();

void undoclicked();

void getInfo();

private:

Estate\* processForm();

void fillForm(Estate \*value);

void showCost(Estate \*value);

Ui::Widget \*ui;

States info;

};

#endif // WIDGET\_H

Widget.cpp

#include "widget.h"

#include "ui\_widget.h"

//переменные ддя хранения текущих данных с формы

int age=-1;//возраст

int clas=0;//индекс класса жилья

QString Name="";//имя

int num=-1;//количество жильцов

double ploshad=-1;//площадь

int srok=0;//индекс срока страхования

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

ui->setupUi(this);

ui->undo->setEnabled(false);

//коннекты

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

connect(ui->age,SIGNAL(textChanged(QString)),this,SLOT(agecheck()));

connect(ui->bvkvadrate,SIGNAL(textChanged(QString)),this,SLOT(ploscheck()));

connect(ui->classlife,SIGNAL(currentIndexChanged(int)),this,SLOT(lifecheck()));

connect(ui->srok,SIGNAL(currentIndexChanged(int)),this,SLOT(srokcheck()));

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

connect(ui->vivod,SIGNAL(clicked(bool)),this,SLOT(getInfo()));

connect(ui->numoflive,SIGNAL(textChanged(QString)),this,SLOT(numcheck()));

connect(ui->undo, SIGNAL(clicked(bool)),this,SLOT(undoclicked()));

}

Widget::~Widget()

{

delete ui;

}

void Widget::getInfo()//проверка на то, заполнены ли все поля

{

if ((age!=1)&&(clas!=0)&&(Name!="")&&(num!=-1)&&(ploshad!=-1)&&(srok!=0))

{

ui->Warning->clear();

processForm();//вызов функции для добавления данных в список и вывода стоимости

}

else

ui->Warning->setText("Не все поля заполнены!");

}

void Widget::undoclicked()//при нажатии кнопки последнего действия

{

info.undo();//вызов функции, возвращающей данные от последнего сохранённого рассчёта

}

void Widget::numcheck()//проверка на число для числа жильцов

{

if (ui->numoflive->text()!="0")//если не 0

if (!(ui->numoflive->text().toInt()))//Проверка на число типа инт

{

ui->Warning->setText("В поле числа жильцов введено не число!");

num=-1;

}

else

{

if ((ui->numoflive->text().toInt()<1)||(ui->numoflive->text().toInt()>=8))//проверка на количество жильцов от 1 до 7

{

ui->Warning->setText("Такое число жильцов недопустимо!");

num=-1;

}

else

{

ui->Warning->clear();

num=ui->numoflive->text().toInt();

}

}

else

{

ui->Warning->setText("Такое количество жильцов недопустимо!");

num=-1;

}

task();

}

void Widget::agecheck()

{

if (ui->age->text()!="0")//если не 0

if (!(ui->age->text().toInt()))//Проверка на число типа инт

{

ui->Warning->setText("В поле возраста введено не число!");

age=-1;

}

else

{

if ((ui->age->text().toInt()<0)||(ui->age->text().toInt()>=100))//проверка на возраст от 0 до 100

{

ui->Warning->setText("Такой возраст недопустим!");

age=-1;

}

else

{

ui->Warning->clear();

age=ui->age->text().toInt();

}

}

else

{

ui->Warning->clear();

age=ui->age->text().toInt();

}

task();

}

void Widget::namecheck()

{

if (ui->name->text().length()<2)

{

ui->Warning->setText("Длинна имени не может быть меньше 2!");

Name="";

}

else

{

if (ui->name->text().contains(QRegularExpression("[^А-Яа-яёЁ ]")))//выполняет проверку на отсутсвие символов кроме русских и пробела

{

ui->Warning->setText("В введённом имени присутсвуют символы,\nкоторые не могут тут быть!");//выполняется если проверка не прошла

Name="";

}

else

{

ui->Warning->clear();//выполняется если проверка прошла

Name=ui->name->text();

}

}

task();

}

void Widget::update()

{

ui->Warning->clear();//очистка поля для ошибок, т.к. теперь всё корректно

auto value = info.getActualData();//получение актуальных данных из списка элементов Estate

if(value != nullptr)

{

fillForm(value);//заполнение формы этими данными

}

ui->undo->setEnabled(info.hasStates());//кнопка последнего действия доступна, если в списке есть элементы

value = nullptr;//текущий элемент пуст

}

Estate\* Widget::processForm()

{

Estate\* elem=new Estate();//создание нового элемента

elem->getall(clas, age ,ploshad,num,srok,Name);//заполнение его данными с формы

info.add(elem);//добавление данного элемента в список

return elem;

}

void Widget::fillForm(Estate \*value)//функция для заполнения всех полей данными

{

switch (value->getType()) {//для класса жилья

case Estate::NOTHING:

clas=0;

break;

case Estate::ECONOM:

clas=1;

break;

case Estate::LUXURIOUS:

clas=2;

break;

case Estate::TOWN\_HOUSE:

clas=3;

break;

case Estate::COTTAGE:

clas=4;

break;

default:

clas=0;

break;

}

ui->classlife->setCurrentIndex(clas);//установка индекса, соответсвующего данному классу жилья

ui->age->setText(QString::number(value->getAge()));//заполнение возраста

ui->bvkvadrate->setText(QString::number(value->getArea()));//заполнение площади

ui->numoflive->setText(QString::number(value->getResidents()));//заполнение числа жильцов

switch (value->getMonths())//для срока страхования и установка соответствующего идекса

{

case -1:

srok=-1;

ui->srok->setCurrentIndex(0);

break;

case 6:

srok=1;

ui->srok->setCurrentIndex(1);

break;

case 12:

srok=2;

ui->srok->setCurrentIndex(2);

break;

case 18:

srok=3;

ui->srok->setCurrentIndex(3);

break;

default:

srok=-1;

ui->srok->setCurrentIndex(0);

break;

}

ui->name->setText(value->getOwner());//заполнение поля с именем

showCost(value);//вывод стоимости

}

void Widget::showCost(Estate \*value)//функция для вывода стоимости

{

ui->cost->setText("Стоимость страхового взноса: "+QString::number(CalculationFacade::getCost(value)));//вызов функции которая рассчитывает стоимость с помощью класса CalculationFacade

}

void Widget::task()//Очищение поля со стоимостью

{

ui->cost->setText("Стоимость страхового взноса:");

}

void Widget::srokcheck()//сохранение данных о текущем состоянии срока

{

srok=ui->srok->currentIndex();

task();//очищение поля стоимости

}

void Widget::lifecheck()

{

clas=ui->classlife->currentIndex();//сохранение данных о текущем выбранном классе жилья

task();//очищение поля стоимости

}

void Widget::ploscheck()//функция для проверки на число

{

if (ui->bvkvadrate->text()!="0")//если не 0

if (!(ui->bvkvadrate->text().toDouble()))//Проверка на число типа дабл

{

ui->Warning->setText("В поле площади введено не число!");

ploshad=-1;

}

else

{

if ((ui->bvkvadrate->text().toDouble()<10)||(ui->bvkvadrate->text().toDouble()>250))//проверка на площадь от 10 до 250

{

ui->Warning->setText("Такая площадь недопустима!");

ploshad=-1;

}

else

{

ui->Warning->clear();

ploshad=ui->bvkvadrate->text().toDouble();

}

}

else

{

ui->Warning->setText("Такая площадь недопустима!");

ploshad=-1;

}

task();

}

Main.cpp

#include "widget.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

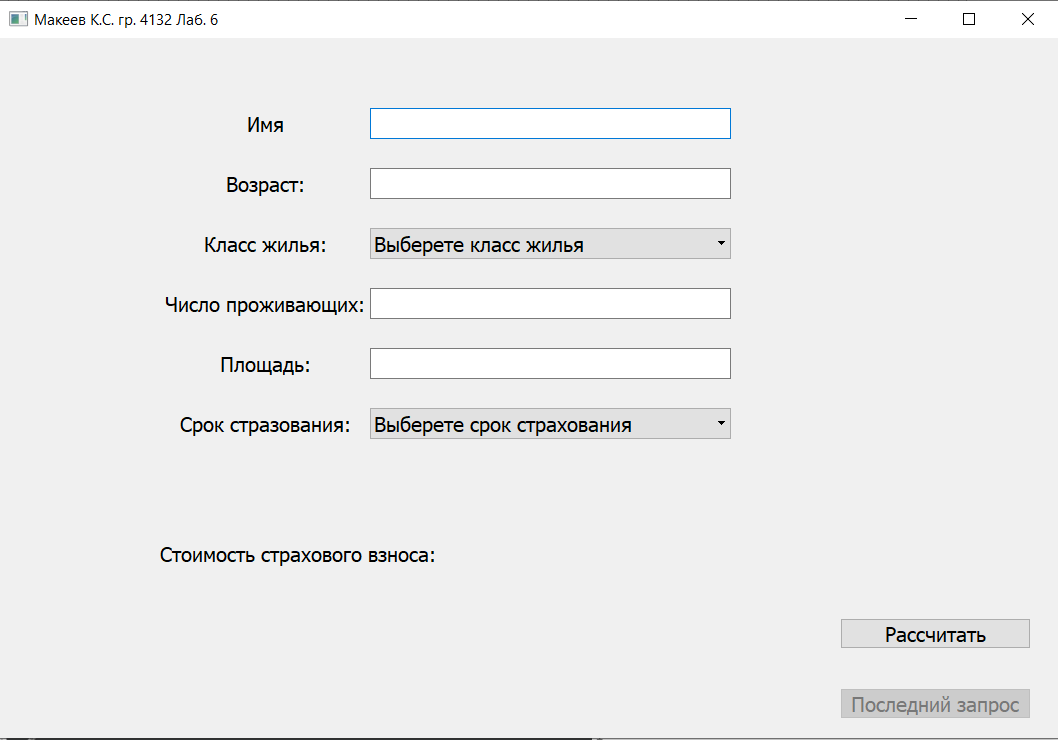
Widget w;

w.show();

return a.exec();

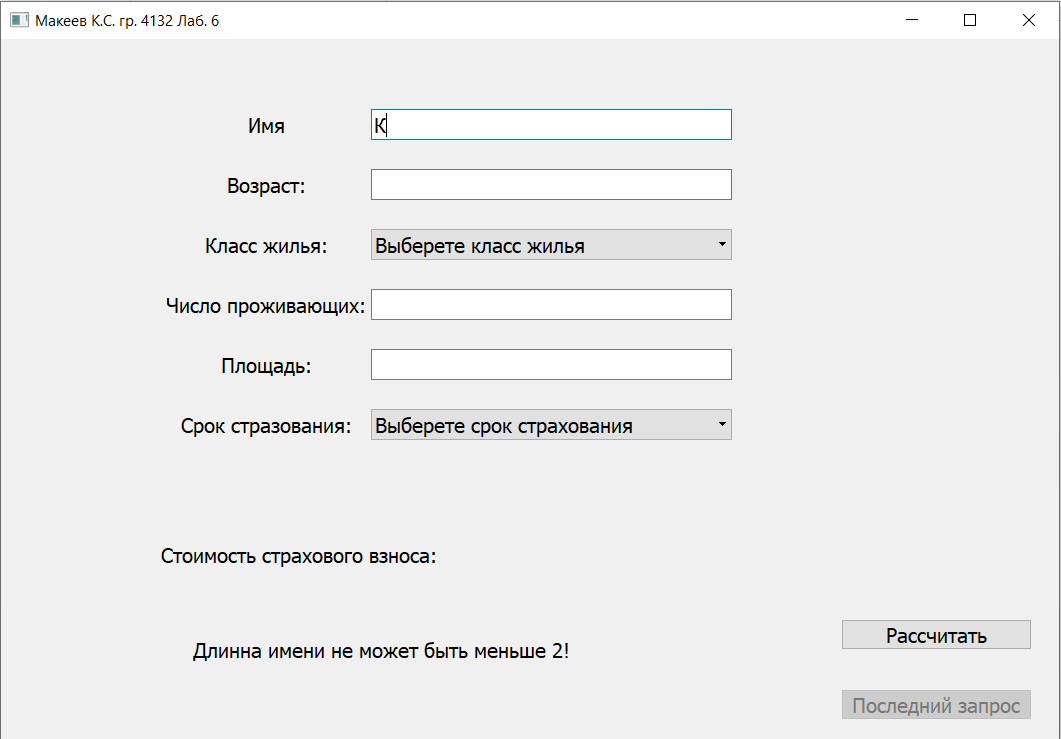
}

**Вид программы:**

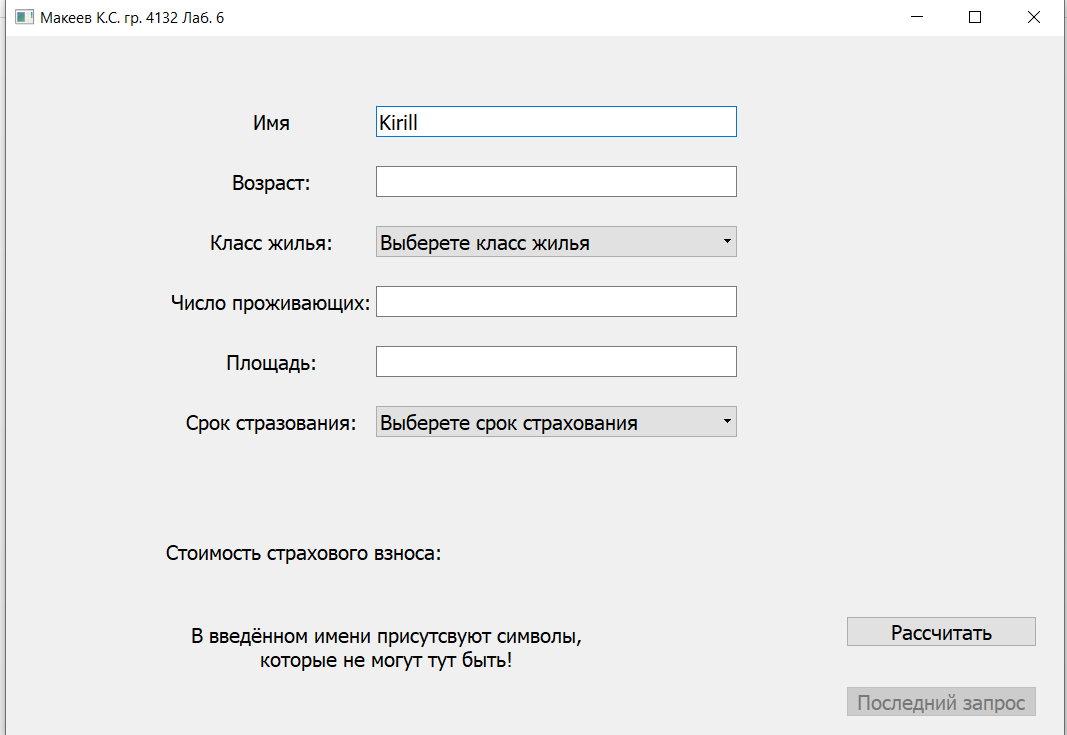


**Пример выполнения функций:**

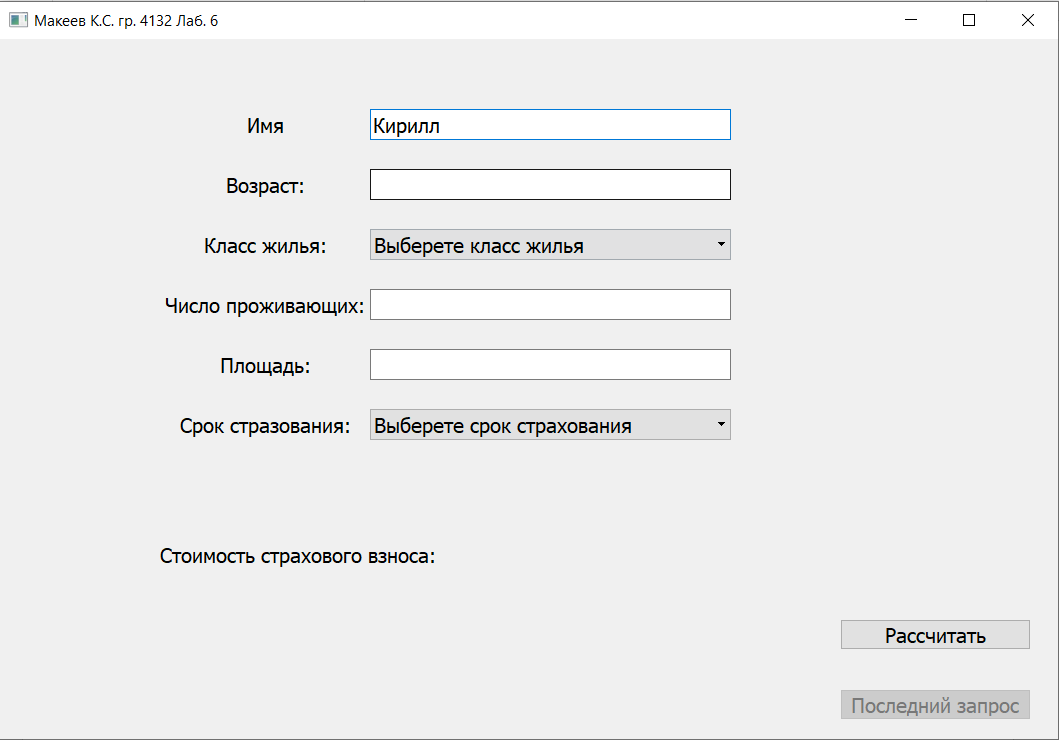
1. Ввод имени длинной менее 2



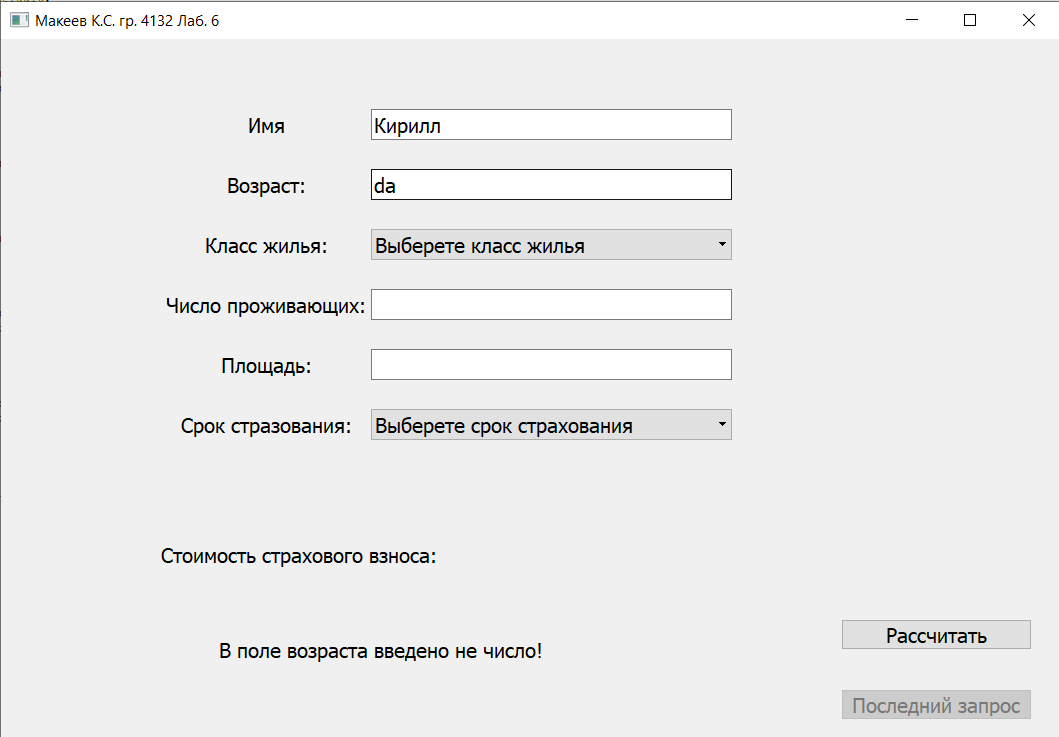
1. Ввод в поле имени не русских символов и пробела



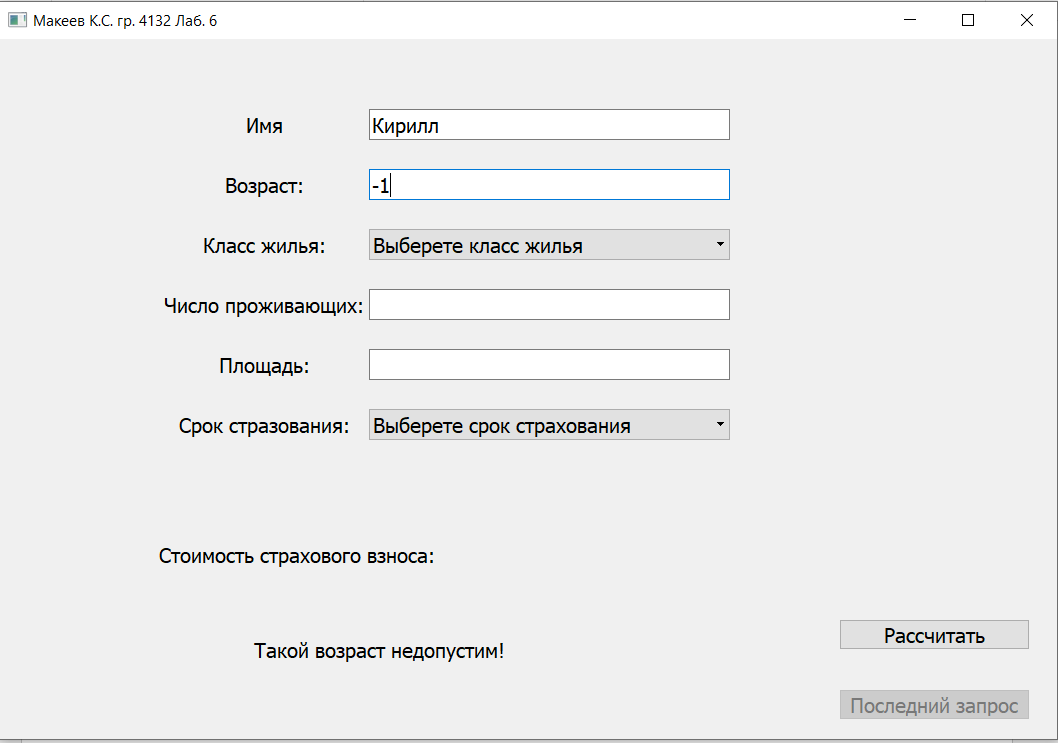
1. Ввод корректного имени



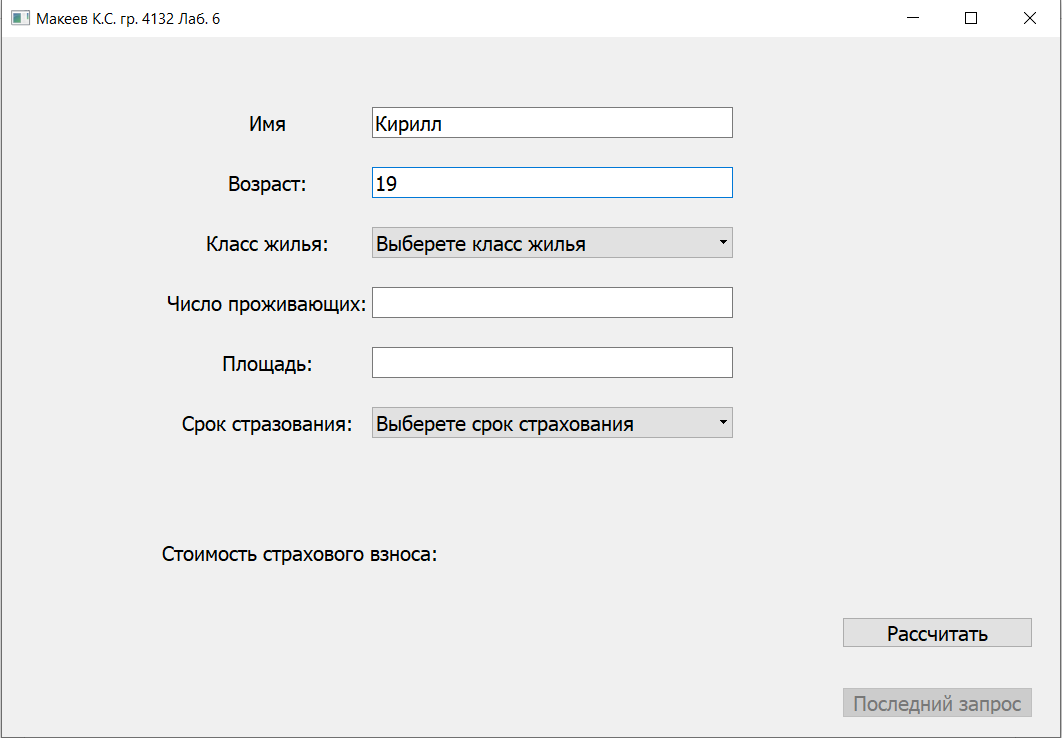
1. Ввод в поле возраста не числа



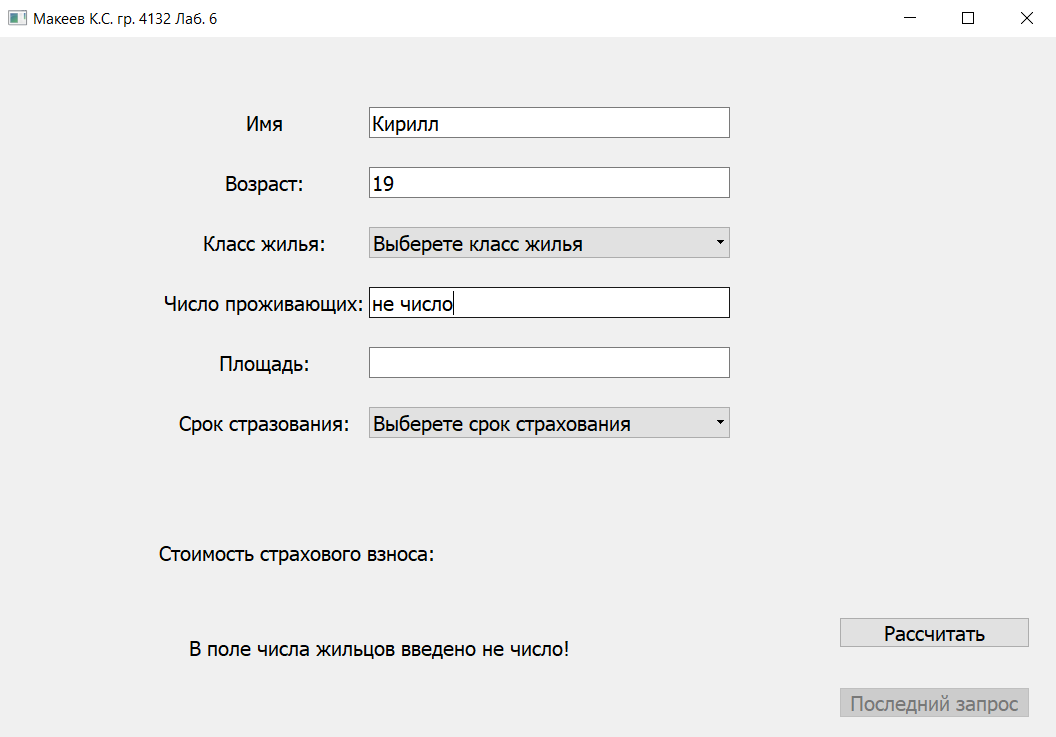
1. Ввод в поле возраста числа меньше 0 или больше 100



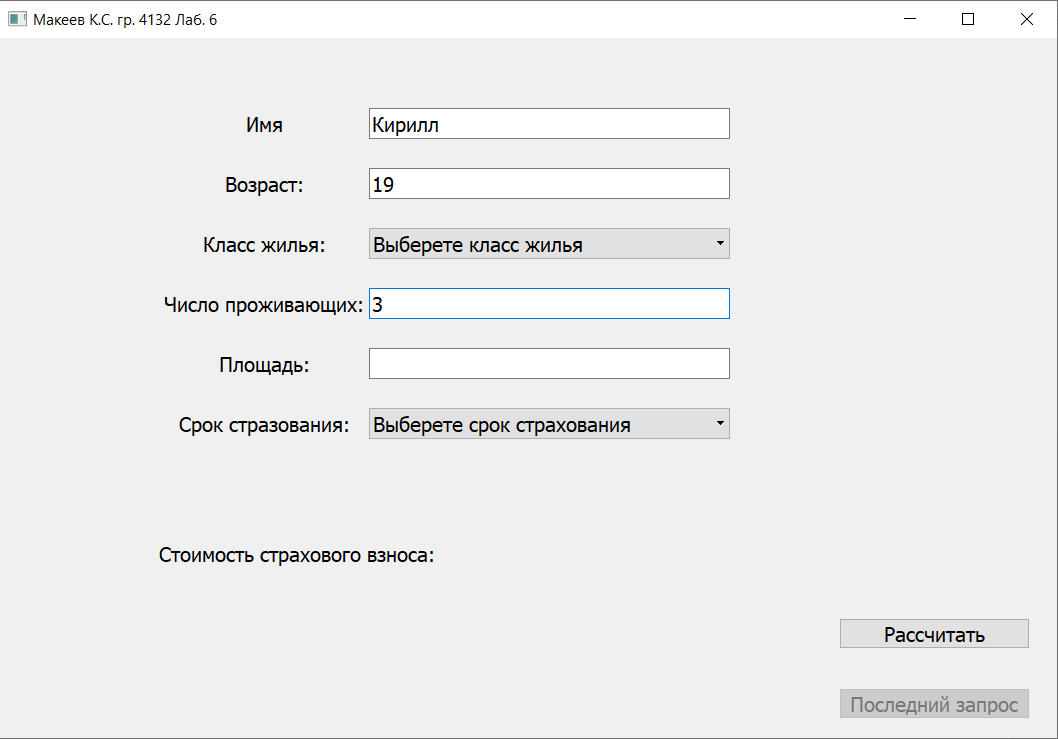
1. Ввод корректного возраста



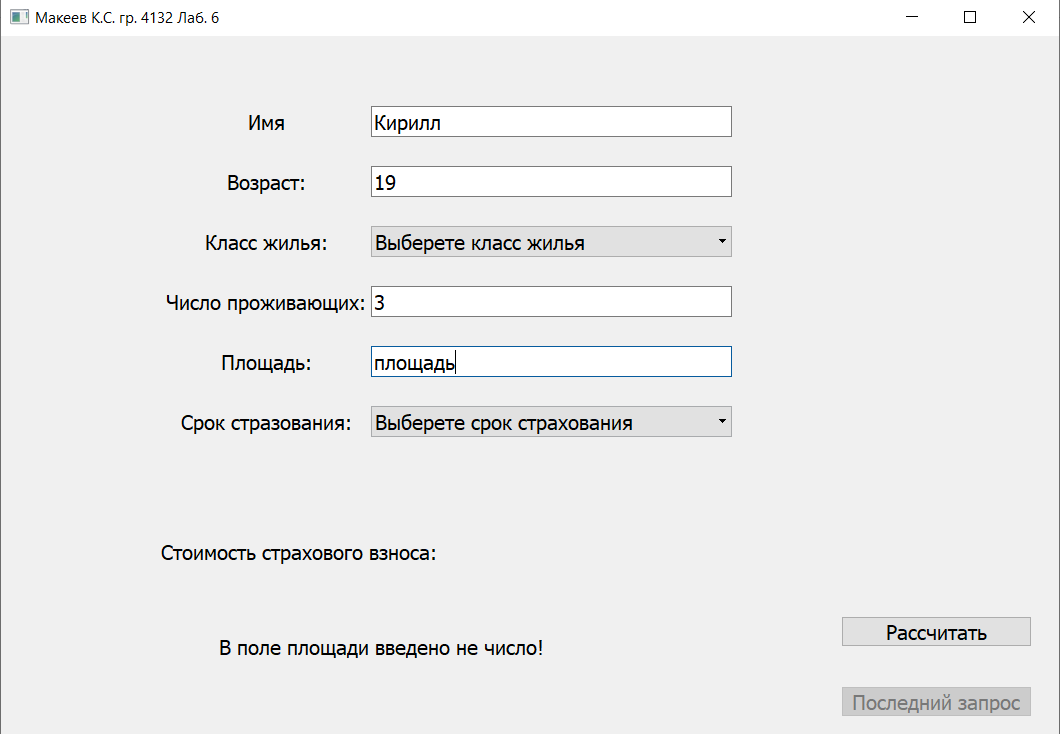
1. Ввод в поле числа жильцов не числа



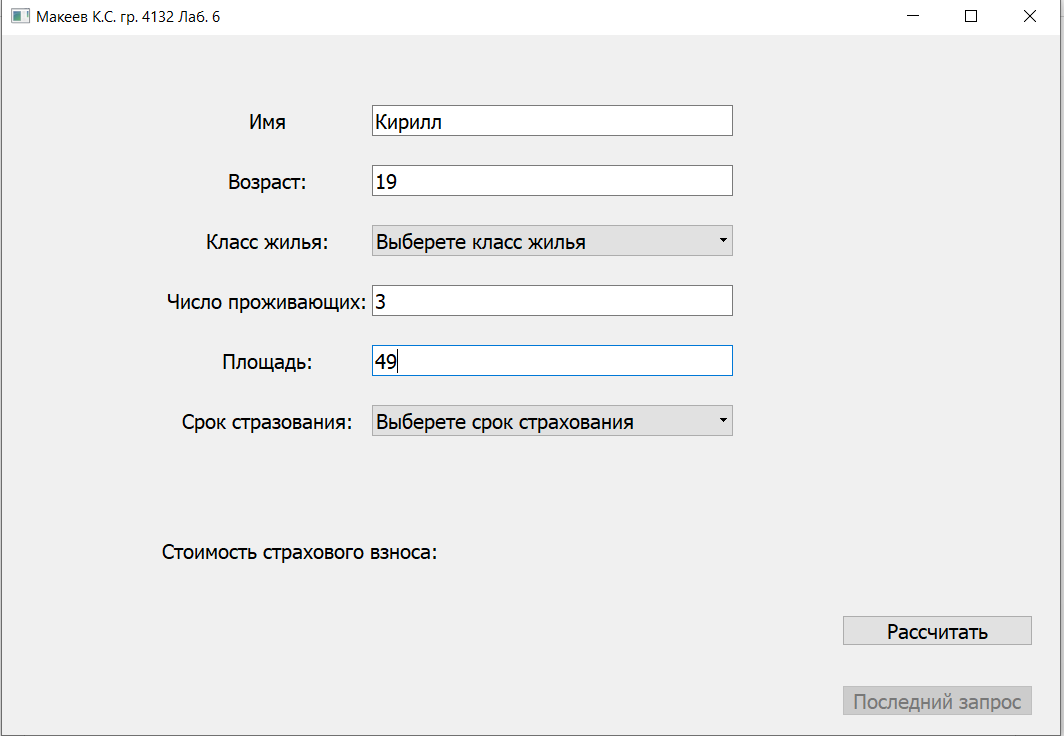
1. Ввод корректного количества жильцов



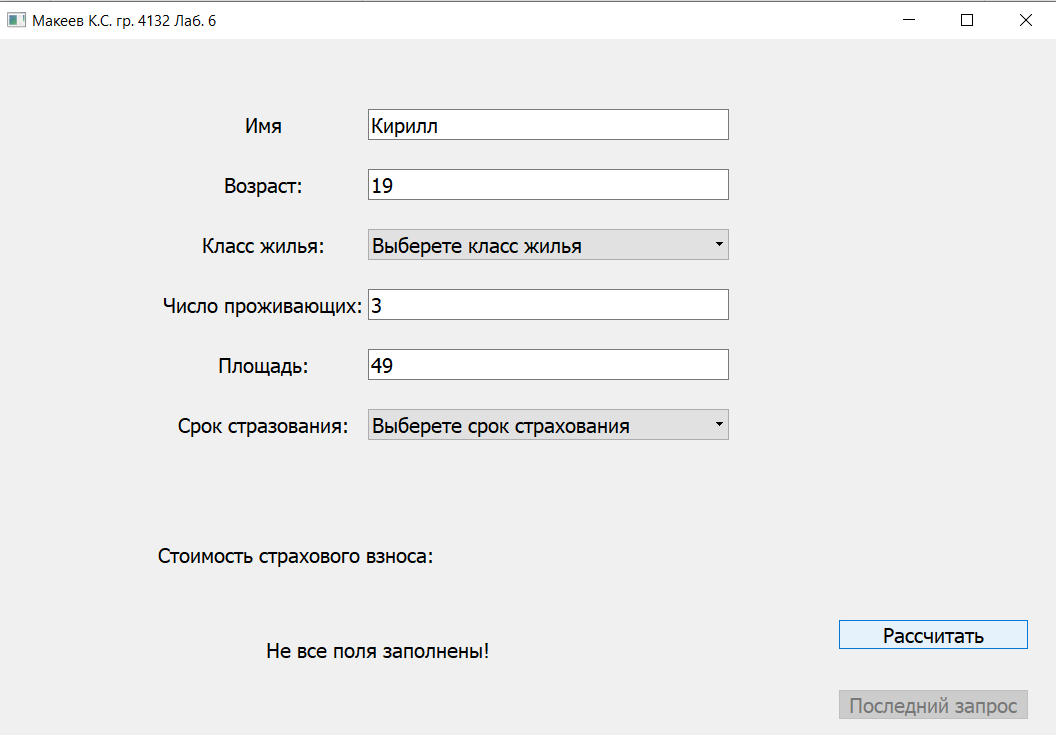
1. Ввод в поле площади не числа



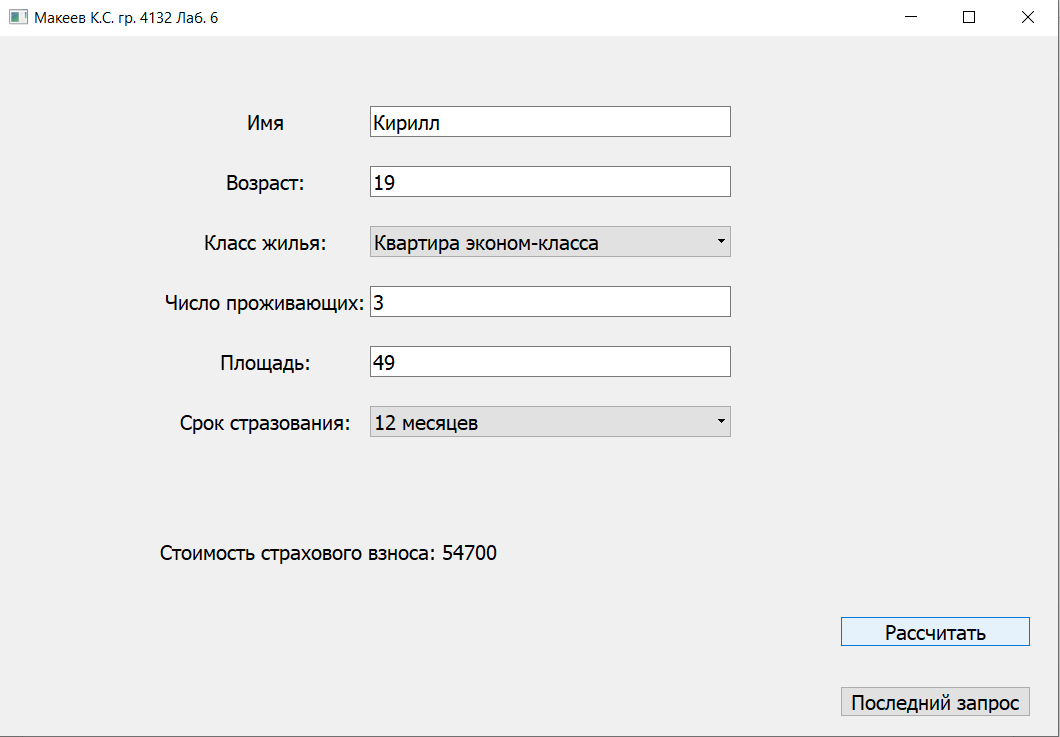
1. Ввод в поле площади корректного значения



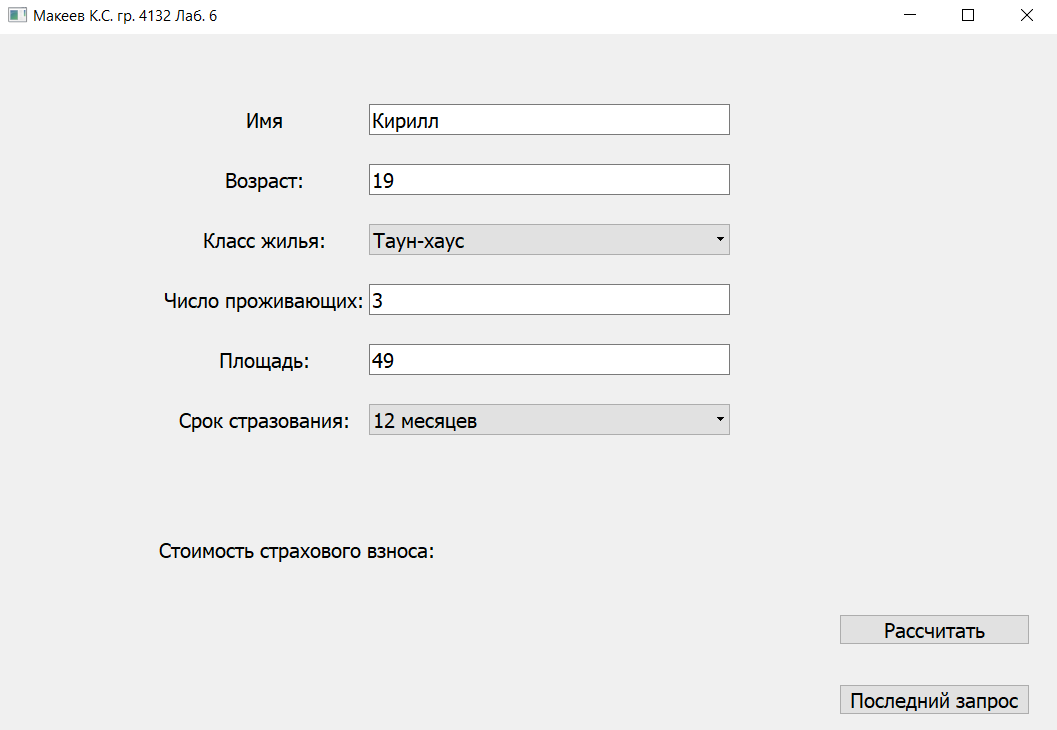
1. Попытка рассчитать стоимость при не полностью заполненной форме



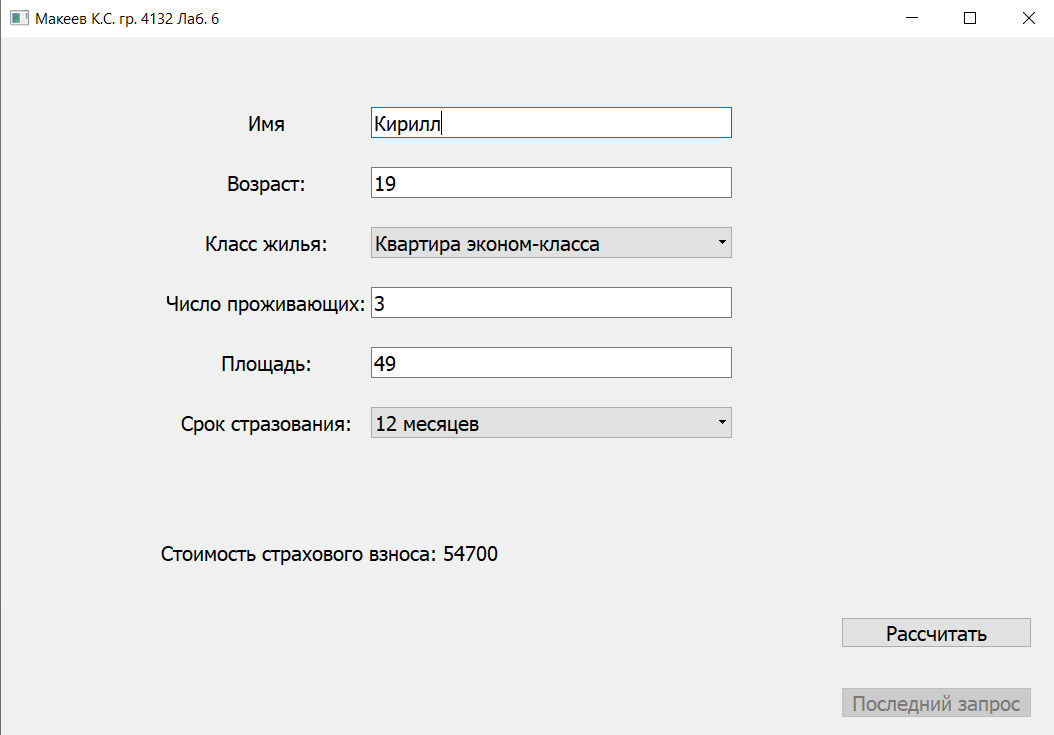
1. Попытка расчёта при полностью заполненной форме



1. При изменении данных очистка стоимости взноса



1. После изменения данных нажат “Последний запрос”



**Вывод:** Изучил принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применил на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Научился использовать шаблон проектирования Наблюдатель и Фасад.