ГУАП

КАФЕДРА № 43

ОТЧЕТ ЗАЩИЩЕН С ОЦЕ	нкой		
ПРЕПОДАВАТЕЛЬ			
доц., канд. техн	ı. наук		Е. О. Шумова
должность, уч. степе	нь, звание	подпись, дата	инициалы, фамилия
	ОТЧЕТ О ЛАБ	БОРАТОРНОЙ РАБ	OTE №8
	Порождающи	ие шаблоны проекти	рования
по курсу: ОБЪЕ	КТНО-ОРИЕНТ	ИРОВАННОМУ ПІ	РОГРАММИРОВАНИЮ
РАБОТУ ВЫПОЛН	ил		
СТУДЕНТ ГР. №	4033	подпись, дата	X.В.Сидиропуло инициалы, фамилия

Лабораторная работа № 8 «Порождающие шаблоны проектирования»

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.

Закрепить знания по теме: Структурные, поведенческие и порождающие шаблоны проектирования.

Задание на лабораторную работу:

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.

1. Листинг программы

main.cpp

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
    QApplication a (argc, argv);
   MainWindow 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),
    info(this)
    ui->setupUi(this);
   ui->btnUndo->setEnabled(false);
 // регистрация слушателя
    connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));
    connect(ui->btnCalc, SIGNAL(pressed()), this, SLOT(btnCalcPressed()));
    connect(ui->btnUndo, SIGNAL(pressed()), this, SLOT(btnUndoPressed()));
   ui->area->setValidator(new QIntValidator(0, 100000, this));
     ui->residents->setValidator(new QIntValidator(0, 100, this));
     ui->age->setValidator(new QIntValidator(0, 100, this));
      ui->owner->setInputMask("AAAAAAAAA");
Widget::~Widget()
   delete ui;
// public slots
void Widget::update()
    auto value = info.getActualData();
    if(value != nullptr){
        fillForm(value);
       showCost(value);
 // update btnUndo state
     ui->btnUndo->setEnabled(info.hasStates());
 // setting value to NULL
    value = nullptr;
// private slots
void Widget::btnCalcPressed()
{
     auto value = processForm();
     showCost(value);
     info.add(value);
     ui->btnUndo->setEnabled(true);
 // setting value to NULL
```

```
value = nullptr;
}
void Widget::btnUndoPressed()
{
     info.undo();
// private
Estate *Widget::processForm()
    //считаваем с формы информацию в объект Estate
    Estate* object = new Estate;
    object->setAge(ui->age->text().toInt());
        object->setOwner(ui->owner->text());
        object->setArea(ui->area->text().toInt());
        object->setRes(ui->residents->text().toInt());
        int t = ui->estateType->currentIndex();
        if(t==0)
            object->setType (Estate::EstateType::ECONOM);
        else if(t==1)
           object->setType(Estate::EstateType::LUXURIOUS);
        else if (t==2)
            object->setType(Estate::EstateType::TOWN HOUSE);
        else if (t==3)
            object->setType(Estate::EstateType::COTTAGE);
        int term = ui->period->currentIndex();
        object->setMonth(term);
        //возвращаем наполненный объект
    return object;
void Widget::fillForm(Estate *value)
    //выводим информацию на форму
    ui->owner->setText(value->getOwner());
        ui->age->setText(QString::number((value->getAge())));
        ui->residents->setText(QString::number(value->getRes()));
        ui->area->setText(QString::number((value->getArea())));
        Estate::EstateType t = value->getType();
        switch (t) {
        case 0:
        ui->estateType->setCurrentIndex(0);
            ui->estateType->setCurrentIndex(1);
        case 2:
             ui->estateType->setCurrentIndex(2);
        case 3:
             ui->estateType->setCurrentIndex(3);
        int term = value->getMonth();
        if (term==6)
            ui->period->setCurrentIndex(0);
        else if(term==12)
           ui->period->setCurrentIndex(1);
        else if(term==18)
            ui->period->setCurrentIndex(2);
void Widget::showCost(Estate *value)
    //создаем объект класса для рассчета стоимости
    CalculationFacade calc;
    //рассчет
    int cost = calc.getCost(value);
```

```
ui->cost->setText(QString::number(cost));
}
void Widget::on_area_textChanged()
    ui->cost->clear();
void Widget::on_period_currentIndexChanged()
    ui->cost->clear();
}
void Widget::on estateType currentIndexChanged()
    ui->cost->clear();
      }
     Townehouse.cpp
#include "townhousecalc.h"
int TownhouseCalc::getCost(Estate* value) {
    return 35*value->getArea()*value->getMonth();
      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 cleare: arra
    qDeleteAll(array);
    array.clear();
void States::undo() {
    array.pop back();
    if (array.isEmpty()) {
    actualData = NULL;
    }
    else
    actualData = array.back();
    //вызов сигнала
    emit notifyObservers();
}
```

```
bool States:: hasStates() {
   if(array.empty())
       return false;
   else
       return true;
}
Estate * States::getActualData() {
  return actualData;
}
void States::add(Estate *value) {
   array.append(value);
Townfactory.cpp
#include "townfactory.h"
#include "townhousecalc.h"
#include "bstractcalc.h"
bstractCalc* TownFactory:: Create() {
   return new TownhouseCalc;
}
Luxuriousapartmentcalc.cpp
#include "luxuriousapartmentcalc.h"
int LuxuriousApartmentCalc::getCost(Estate * value) {
   return 25*value->getArea()*value->getMonth();
}
Luxurfactory.cpp
#include "luxurfactory.h"
#include "luxuriousapartmentcalc.h"
#include "bstractcalc.h"
bstractCalc* LuxurFactory:: Create(){
    return new LuxuriousApartmentCalc;
Estate.cpp
#include "estate.h"
Estate::Estate(QObject *parent) : QObject(parent)
   age = 19;
    area = 170;
   residents = 5;
   months = 6;
    type = ECONOM;
   owner = "XeTar";
Estate::EstateType Estate::getType() const{
   return type;
int Estate::getAge() const{
```

```
return age;
int Estate::getArea() const{
      return area;
int Estate::getRes() const{
       return residents;
int Estate::getMonth() const{
    return months;
QString Estate::getOwner() const{
       return owner;
    }
void Estate::setType (EstateType t) {
   type = t;
}
void Estate::setAge(int a) {
  age = a;
}
void Estate::setArea(int a) {
   area = a;
}
void Estate::setRes(int r) {
   residents = r;
void Estate::setMonth(int i) {
   if(i==0)
       months = 6;
    else if(i==1)
      months = 12;
    else if(i==2)
       months = 18;
void Estate::setOwner(QString str) {
   owner = str;
Cottagefactory.cpp
#include "cottagefactory.h"
#include "cottagecalc.h"
#include "bstractcalc.h"
bstractCalc* CottageFactory:: Create() {
   return new CottageCalc;
Cottagecalc.cpp
#include "cottagecalc.h"
int CottageCalc::getCost(Estate* value) {
   return 20*value->getArea()*value->getMonth();
}
```

Calculationfacede.cpp

```
#include "calculationfacade.h"
#include "apartfactory.h"
#include "luxurfactory.h"
#include "cottagefactory.h"
#include "townfactory.h"
#include "bstractcalc.h"
#include "apartmentcalc.h"
#include "luxuriousapartmentcalc.h"
#include "townhousecalc.h"
CalculationFacade::CalculationFacade(QObject *parent) : QObject(parent)
int CalculationFacade::getCost(Estate *value)
 int cost;
switch (value->getType()) {
   case Estate::EstateType::ECONOM:
    ApartFactory* a = new ApartFactory;
    bstractCalc* b = a->Create();
     cost = b->getCost(value);
    break;
 }
   case Estate::EstateType::LUXURIOUS:
 {
    LuxurFactory* a = new LuxurFactory;
    bstractCalc* b = a->Create();
    cost = b->getCost(value);
       break;
 }
   case Estate::EstateType::TOWN HOUSE:
 {
    TownFactory* a = new TownFactory;
    bstractCalc* b = a->Create();
    cost = b->getCost(value);
       break;
 }
   case Estate::EstateType::COTTAGE:
 {
    CottageFactory* a = new CottageFactory;
    bstractCalc* b = a->Create();
    cost = b->getCost(value);
       break;
 }
    default:
       cost = -1;
       break;
 }
 return cost;
Apartamentcalc.cpp
#include "apartmentcalc.h"
int ApartmentCalc::getCost(Estate *value) {
```

```
return 10*value->getMonth()*value->getArea();
}

Apartfactory.cpp

#include "apartfactory.h"
#include "apartmentcalc.h"
#include "bstractcalc.h"

bstractCalc* ApartFactory:: Create() {
    return new ApartmentCalc;
}
```

Widget.h

```
#ifndef WIDGET H
#define WIDGET H
#include <QWidget>
#include <states.h>
#include <estate.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 btnCalcPressed();
    void btnUndoPressed();
    void on area textChanged();
    void on period currentIndexChanged();
    void on estateType currentIndexChanged();
private:
    Estate *processForm();
    void fillForm(Estate *value);
    void showCost(Estate *value);
private:
    Ui::Widget *ui;
    States info;
};
#endif // WIDGET_H
```

Townhousecalc.h

```
#ifndef TOWNHOUSECALC H
#define TOWNHOUSECALC H
#include <estate.h>
#include <bstractcalc.h>
class TownhouseCalc : public QObject, public bstractCalc
public:
   int getCost(Estate* value);
#endif // TOWNHOUSECALC H
Townfacroty.h
#ifndef TOWNFACTORY H
#define TOWNFACTORY H
#include <calcfactory.h>
#include<bstractcalc.h>
class TownFactory:public CalcFactory
{
public:
   bstractCalc* Create();
#endif // TOWNFACTORY H
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);
signals:
    void notifyObservers();
private:
    QList<Estate *> array;
    Estate *actualData;
Luxuriousapartmentcalc.h
#ifndef LUXURIOUSAPARTMENTCALC H
#define LUXURIOUSAPARTMENTCALC H
#include <estate.h>
#include<bstractcalc.h>
class LuxuriousApartmentCalc : public QObject, public bstractCalc
```

```
public:
    int getCost(Estate * value);
};
#endif // LUXURIOUSAPARTMENTCALC H
```

```
Luxurfacrtory.h
#ifndef LUXURFACTORY H
#define LUXURFACTORY H
#include <calcfactory.h>
#include<bstractcalc.h>
class LuxurFactory:public CalcFactory
public:
   bstractCalc* Create();
};
#endif // LUXURFACTORY 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);
    EstateType getType() const;
    int getAge()const;
    int getArea()const;
    int getRes()const;
    int getMonth()const;
    QString getOwner()const;
    void setType(EstateType);
    void setAge(int);
    void setArea(int);
    void setRes(int);
    void setMonth(int);
    void setOwner(QString);
private:
    int age;
    int area;
    int residents;
    int months;
    EstateType type;
    QString owner;
```

```
};
#endif // ESTATE H
Cottagecalc.h
#ifndef COTTAGECALC H
#define COTTAGECALC H
#include <estate.h>
#include <bstractcalc.h>
class CottageCalc : public bstractCalc
public:
    int getCost(Estate* value);
#endif // COTTAGECALC H
Cottagefactory.h
#ifndef COTTAGEFACTORY H
#define COTTAGEFACTORY H
#include <calcfactory.h>
#include<bstractcalc.h>
class CottageFactory:public CalcFactory
public:
   bstractCalc* Create();
#endif // COTTAGEFACTORY H
Calculationfacede.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
Calcfactory.h
#ifndef CALCFACTORY H
#define CALCFACTORY H
#include<bstractcalc.h>
```

class CalcFactory

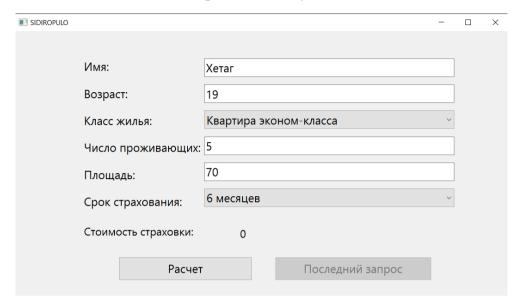
```
{
public:
    virtual bstractCalc* Create()=0;
   virtual ~CalcFactory() { };
};
#endif // CALCFACTORY H
Bstractcalc.h
#ifndef BSTRACTCALC H
#define BSTRACTCALC H
#include <estate.h>
class bstractCalc
{
public:
    virtual int getCost(Estate *)=0;
    virtual ~bstractCalc(){};
#endif // BSTRACTCALC H
Apartfactory.h
#ifndef APARTFACTORY H
#define APARTFACTORY H
#include <calcfactory.h>
#include<bstractcalc.h>
class ApartFactory:public CalcFactory
{
public:
  bstractCalc* Create();
#endif // APARTFACTORY H
Apartamentcalc.h
#ifndef APARTMENTCALC H
#define APARTMENTCALC H
#include<estate.h>
#include<bstractcalc.h>
class ApartmentCalc : public QObject, public bstractCalc
public:
   int getCost(Estate *value);
};
```

#endif // APARTMENTCALC_H

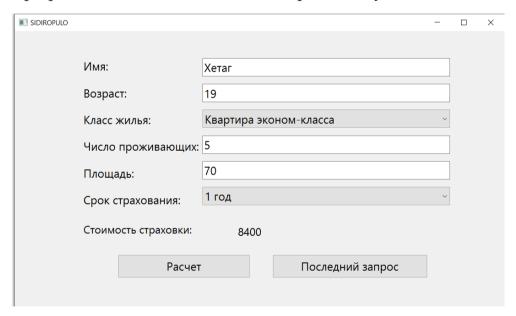
3. Работа программы

Расчет стоимости страхования:

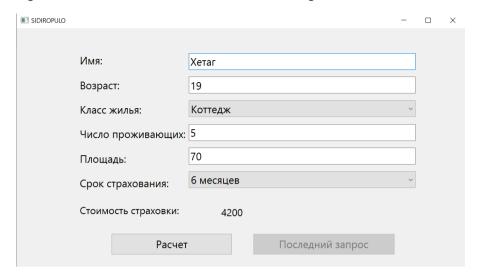
Кнопка «Последний запрос» недоступна



При расчете кнопка «последний запрос» доступна



При нажатии кнопка «последний запрос» снова становится недоступна



Вывод.

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