

ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

доц., канд. техн. наук  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Е. О. Шумова  
\_\_\_\_\_  
инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

Порождающие шаблоны проектирования

по курсу: ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4033

\_\_\_\_\_  
подпись, дата

Х.В.Сидиропуло  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2022

## **Лабораторная работа № 8 «Порождающие шаблоны проектирования»**

**Цель работы:** Изучить принципы построения приложений с графическим интерфейсом, используя 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("AAAAAAAAAA");
}

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()
    {
        //считываем 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);
            case 1:
                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 расчета стоимости
        CalculationFacade calc;
        //расчет
        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 = "Xerar";
}

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();
}

```

## Calculationfacade.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;
}
```

## Apartmentcalc.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 <bstratcalc.h>

class TownhouseCalc : public QObject, public bstratCalc
{
public:
    int getCost(Estate* value);
};
#endif // TOWNHOUSECALC_H
```

## Townfactory.h

```
#ifndef TOWNFACTORY_H
#define TOWNFACTORY_H

#include <calcfactory.h>
#include <bstratcalc.h>

class TownFactory:public CalcFactory
{
public:
    bstratCalc* 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 <bstratcalc.h>

class LuxuriousApartmentCalc : public QObject, public bstratCalc
{
```

```

public:
    int getCost(Estate * value);
};
#endif // LUXURIOUSAPARTMENTCALC_H

```

## Luxurfacrtoary.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 <bstrctcalc.h>

class CottageCalc : public bstrctCalc
{
public:
    int getCost(Estate* value);
};
#endif // COTTAGECALC_H
```

## Cottagefactory.h

```
#ifndef COTTAGEFACTORY_H
#define COTTAGEFACTORY_H
#include <calcfactory.h>
#include <bstrctcalc.h>

class CottageFactory:public CalcFactory
{
public:
    bstrctCalc* 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 <bstrctcalc.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

```

## Apartmentcalc.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

The screenshot shows a window titled "SIDIROPULO" with a form for calculating insurance costs. The form contains the following fields and values:

Field	Value
Имя:	Хетаг
Возраст:	19
Класс жилья:	Квартира эконом-класса
Число проживающих:	5
Площадь:	70
Срок страхования:	6 месяцев
Стоимость страховки:	0

At the bottom of the form, there are two buttons: "Расчет" (Calculation) and "Последний запрос" (Last request). The "Последний запрос" button is disabled (grayed out).

При расчете кнопка «последний запрос» доступна

The screenshot shows the same "SIDIROPULO" window after a calculation. The form fields and values are the same as in the previous screenshot, except for the "Срок страхования" (Insurance term) which is now "1 год" (1 year) and the "Стоимость страховки" (Insurance cost) which is now "8400".

Field	Value
Имя:	Хетаг
Возраст:	19
Класс жилья:	Квартира эконом-класса
Число проживающих:	5
Площадь:	70
Срок страхования:	1 год
Стоимость страховки:	8400

The buttons "Расчет" (Calculation) and "Последний запрос" (Last request) are both enabled (not grayed out).

При нажатии кнопка «последний запрос» снова становится недоступна

SIDIROPULO

Имя:

Хетаг

Возраст:

19

Класс жилья:

Коттедж

Число проживающих:

5

Площадь:

70

Срок страхования:

6 месяцев

Стоимость страховки:

4200

Расчет

Последний запрос

### **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.