

## Лабораторная работа № 5

### Разработка программ с использованием паттернов поведения

*Цель работы:* научиться разрабатывать программы с использованием поведенческих шаблонов проектирования *Стратегия*, *Состояние* и *Шаблонный метод*.

#### 1. Теоретические сведения



##### ***Поведенческие шаблоны проектирования. Диаграммы конечных автоматов UML***

Поведенческие шаблоны (англ. «Behavioral Patterns») предназначены для определения алгоритмов и способов реализации взаимодействия различных объектов и классов.

Одним из средств описания поведения программных объектов в языке UML являются диаграммы конечных автоматов (диаграммы состояний).

Диаграммы конечных автоматов (англ. «state machine diagrams») UML отображают жизненный цикл объекта с помощью состояний, событий и переходов.

Основными элементами диаграмм конечных автоматов являются:

- *состояние* – это ситуация во время жизни объекта, в которой он удовлетворяет определённым условиям, выполняет определённую деятельность или находится в ожидании событий. Состояния изображаются на диаграмме в виде прямоугольников со скруглёнными углами. Специальными видами состояний являются начальное и конечное состояния, изображаемые соответственно символами  и .
- *переход* – это отношение между двумя состояниями, указывающее на то, что объект из первого состояния перейдёт во второе, при выполнении определённого условия. Переходы на диаграммах конечных автоматов изображают стрелками, ведущими от одного состояния к другому;
- *событие* – это значимое или заслуживающее внимания происшествие, которое может инициализировать переход объекта от одного состояния к другому. Событием может являться поступление сигнала, выполнение какого-либо условия, истечение определённого периода времени. События разделяют на внешние, внутренние и временные.

Пример простой диаграммы конечных автоматов для объекта «входная дверь» показан на рисунке 1.

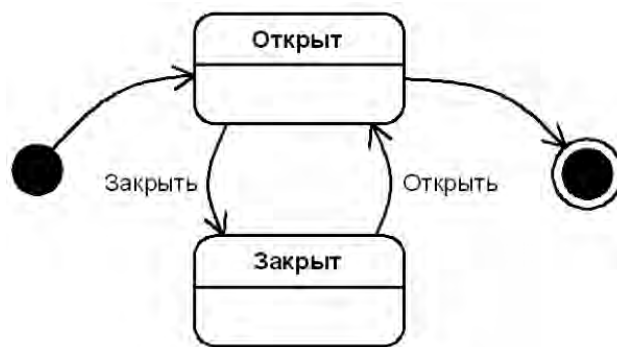


Рисунок 1 – Простая диаграмма конечных автоматов (состояний) UML

## Паттерн Состояние

Шаблон *Состояние* (англ. «State») управляет изменением поведения объекта в зависимости от его внутреннего состояния.

*Проблема.*

Как изменять поведение объекта в зависимости от его внутреннего состояния?

*Решение.*

Определить для каждого состояния отдельный класс со стандартным интерфейсом.

*Структура.*

Диаграмма классов шаблона Состояние представлена на рисунке 2.

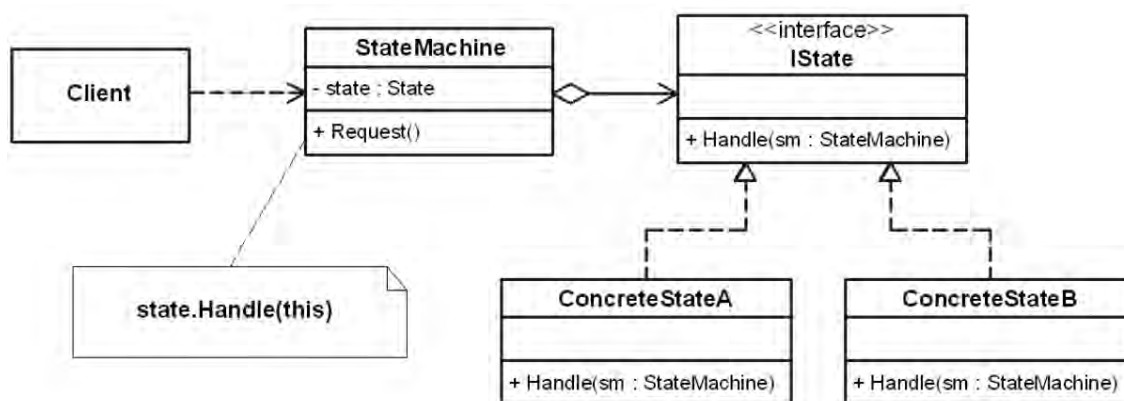


Рисунок 2 – Диаграмма классов шаблона Состояние

*Участники шаблона:*

State – абстрактный класс, определяющий общий интерфейс для всех конкретных состояний;

StateMachine – класс с несколькими внутренними состояниями, хранит экземпляр класса State;

ConcreteStateA, ConcreteStateB – классы конкретных состояний, обрабатывающие запросы от класса StateMachine; каждый класс предоставляет собственную реализацию обработки запроса.

*Реализация.*

Пример реализации паттерна *Состояние* на языке C# приведён ниже.

```
// Представляет состояния
interface IState
{
    void Handle(StateMachine sm);
}
// Представляет конкретные
состояния Aclass
ConcreteStateA : IState
{
    public void Handle(StateMachine sm)
    {
        Console.Write("Объект {0} в
сост. A",
sm.ToString());
    }
}
// Представляет конкретные
состояния Bclass
ConcreteStateB : IState
{
    public void Handle(StateMachine sm)
    {
        Console.Write("Объект {0} в
сост. B",
sm.ToString());
    }
}
// Представляет конечные автоматы
class StateMachine
{
    public IState State {
get; set; }public void
Request()
{
    State.Handle(this);
}
}
// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        StateMachine sm = new
        StateMachine();sm.State = new
        ConcreteStateA();
        sm.Request();
        sm.State = new
        ConcreteStateB();
        sm.Request();
    }
}
```

}  
}

## Паттерны Стратегия и Шаблонный метод

**Паттерн Стратегия** (англ., «Strategy» или «Policy») позволяет менять алгоритм независимо от клиентов, которые его используют.

*Проблема.*

Как спроектировать изменяемые, но надёжные алгоритмы (стратегии)?

*Решение.*

Определить для каждого алгоритма (стратегии) отдельный класс со стандартным интерфейсом.

*Структура.*

Диаграмма классов шаблона Стратегия показана на рисунке 3.

*Участники шаблона:*

Strategy – класс, определяющий общий для всех поддерживаемых алгоритмов интерфейс;

Context – класс, хранящий ссылку на экземпляр класса Стратегия;

ConcreteStrategyA, ConcreteStrategyB – классы, представляющие конкретные стратегии, использующие интерфейс класса Strategy для реализации алгоритма.

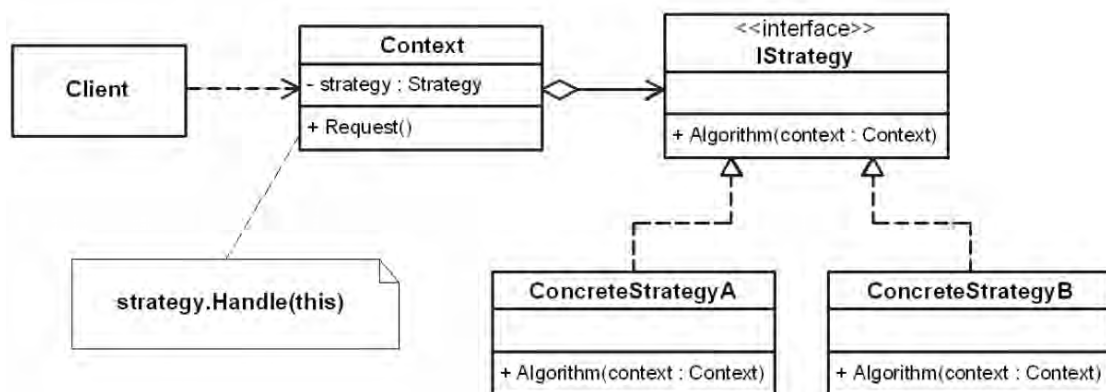


Рисунок 3 – Диаграмма классов шаблона Стратегия

*Реализация.*

Пример реализации шаблона Стратегия на языке C# приведён ниже.

```
// Представляет стратегии, реализуемые классами
public interface IStrategy
{
    void Algorithm();
}
// Представляет конкретные
стратегии A
public class
ConcreteStrategyA : IStrategy
{
    public void Algorithm()
    {
```

```

        Console.WriteLine("Алгоритм стратегии
        A");
    }
}
// Представляет конкретные
стратегии B
public class
ConcreteStrategyB : IStrategy
{
    public void Algorithm()
    {
        Console.WriteLine("Алгоритм стратегии
        B");
    }
}
// Представляет контексты, использующие стратегии
public class Context
{
    public IStrategy Strategy {
    get; set; }
    public void
    Request()
    {
        Strategy.Algorithm();
    }
}
// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        Context context = new Context();
        context.Strategy = new
        ConcreteStrategyA();
        context.Request();
        context.Strategy = new
        ConcreteStrategyB();
        context.Request();
    }
}

```

**Паттерн Шаблонный метод** (англ., «Template Method») определяет основу алгоритма (шаблонный метод) в базовом классе и позволяет производным классам переопределять отдельные шаги алгоритма, не изменяя его структуру в целом.

*Проблема.*

Как определить алгоритм и реализовать возможность переопределения некоторых шагов алгоритма в производных классах без изменения общей структуры алгоритма?

*Решение.*

Создать абстрактный класс, определяющий следующие операции:

- шаблонный метод, который задаёт основу алгоритма;

- абстрактные операции, которые замещаются в производных классах для реализации шагов алгоритма.

*Структура.*

Диаграмма классов паттерна *Шаблонный метод* показана на рисунке 4.

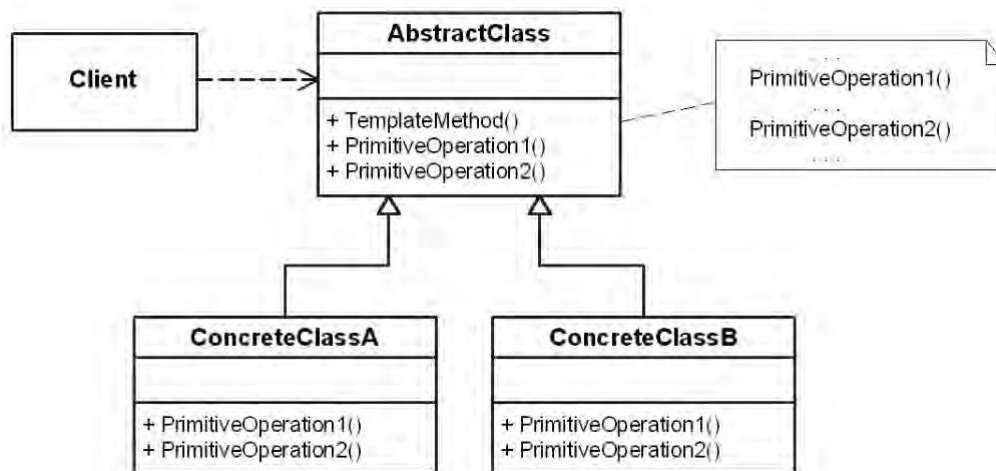


Рисунок 4 – Диаграмма классов шаблона Шаблонный метод

*Участники шаблона:*

AbstractClass определяет абстрактные частные операции и содержит шаблонный метод, который вызывает эти операции;

ConcreteClassA, ConcreteClassB реализуют частные операции, вызываемые шаблонным методом.

*Реализация.*

Пример реализации шаблона Шаблонный метод на языке C# приведён ниже.

```

// Представляет Абстрактный класс
public abstract class AbstractClass
{
    int a = 10;
    // Шаблонный метод
    public int TemplateMethod(int x)
    {
        int y = 3;
        if (x > a) y +=
        PartMethod1(x);else y
        += PartMethod2(x);
        y -=
        a;
        retu
        rn y;
    }
    public abstract int
    PartMethod1(int x);public
    abstract int PartMethod2(int
    x);
}
// Представляет Конкретный класс А

```

```

public class ConcreteClassA : AbstractClass
{
    int k = 5;
    public override int PartMethod1(int x)
    { return k + x; }
    public override int PartMethod2(int x)
    { return k * x; }
}
// Представляет Конкретный класс B
public class ConcreteClassB : AbstractClass
{
    int k = 3;
    public override int PartMethod1(int x)
    { return x * x - k; }
    public override int PartMethod2(int x)
    { return x + x * k; }
}
// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        AbstractClass a = new
        ConcreteClassA(); AbstractClass
        b = new ConcreteClassB(); int x
        = 20;
        Console.WriteLine(a.TemplateMet
        hod(x));
        Console.WriteLine(b.TemplateMet
        hod(x));
    }
}

```

## 2. Задания к лабораторной работе

Для заданного варианта задания разработать UML-диаграмму классов, диаграмму состояний и диаграмму последовательности.

Разработать консольное приложение (C++, C#). Допускается вводить дополнительные понятия предметной области. В программе предусмотреть тестирование функциональности созданных объектов классов.

Отчет по лабораторной работе – файл формата pdf. Формат имени файла отчета: <НомерГруппы >\_<ФамилияСтудента>.pdf. В отчет включить построенные диаграммы и исходный код программы (при необходимости и заголовочные файлы). Формат отчета см. в Приложении. Отчет загрузить в LMS.

**При защите лабораторной работы:** уметь объяснить логику и детали работы программы; реализацию паттернов проектирования на примере разработанной программы.

### Варианты заданий

- 1 Изучить пример проектирования программной системы с использованием паттерна Состояние [Турчин-Архитектура ИС.pdf [Электронный ресурс], с. 143–154].
- 2 Разработать диаграмму конечных автоматов (состояний) для заданного класса (таблица 1). Описать в форме таблицы варианты реакции экземпляра класса на операции, вызываемые в указанных состояниях. Разработать UML-диаграммы классов и диаграммы последовательности.

Таблица 1 – Варианты заданий для разработки приложения с использованием шаблона Состояние

Номер варианта	Классы, их атрибуты и операции	Состояние
1	<i>Телефон</i> <i>Атрибуты:</i> номер, баланс, вероятность поступления звонка. <i>Операции:</i> позвонить, ответить на звонок, завершить разговор, пополнить баланс	Ожидание, Звонок, Разговор, Заблокирован (баланс отрицательный)
2	<i>Банкомат</i> <i>Атрибуты:</i> ID, общая сумма денег в банкомате, вероятность отсутствия связи с банком. <i>Операции:</i> ввести PIN-код, снять заданную сумму, завершить работу, загрузить деньги в банкомат	Ожидание, Аутентификация пользователя, Выполнение операций, Заблокирован (денег нет)
3	<i>Грузовой лифт</i> <i>Атрибуты:</i> текущий этаж, грузоподъемность, вероятность отключения электроэнергии. <i>Операции:</i> вызвать на заданный этаж, загрузить, разгрузить, восстановить подачу энергии	Покой, Движение, Перегружен, Нет питания, Авария

- 3 Разработать библиотеку классов, включающую необходимые классы для реализации паттерна *Состояние* (класс Конечный автомат, интерфейс Состояние, классы Конкретные состояния).
- 4 <Опция на дополнительные баллы>: Разработать приложение Windows Forms для управления состояниями экземпляров класса Конечный автомат. При использовании Windows-форм вместо исходного кода в отчет вставить ссылку на репозиторий GitHub с



проектом.

Распределение вариантов заданий

№ по списку группы	№ варианта	№ по списку группы	№ варианта	№ по списку группы	№ варианта
1	1	11	2	21	3
2	2	12	3	22	1
3	3	13	1	23	2
4	1	14	2	24	3
5	2	15	3	25	1
6	3	16	1	26	2
7	1	17	2	27	3
8	2	18	3	28	1
9	3	19	1	29	2
10	1	20	2	30	3

Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 00

Группа: 000-000                      Студент: Иванов Иван Иванович

Задание на лабораторную работу

<Формулировка                      задания                      согласно                      варианту  
.....  
.....>

Диаграмма классов

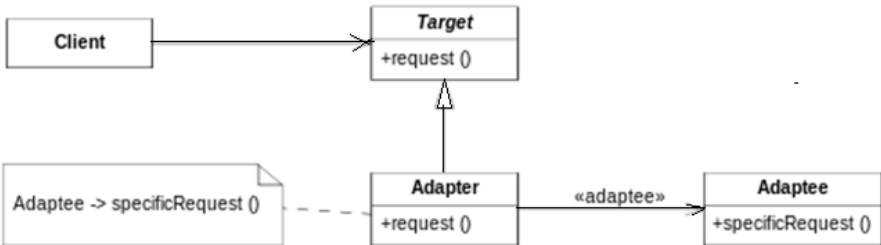
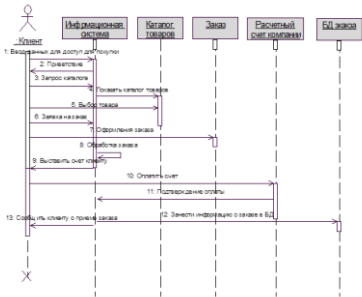


Диаграмма последовательности



Исходный код программы

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../Add/Add.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
    TEST_CLASS(UnitTest1)
    {
    public:

        TEST_METHOD(TestMethod1)
        {
            Assert::AreEqual(2, add(1, 1));
        }
    }
}
```