Лабораторная работа 9

Изучение и практическое применение поведенческого шаблона проектирования Состояние

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

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 рис. 1.

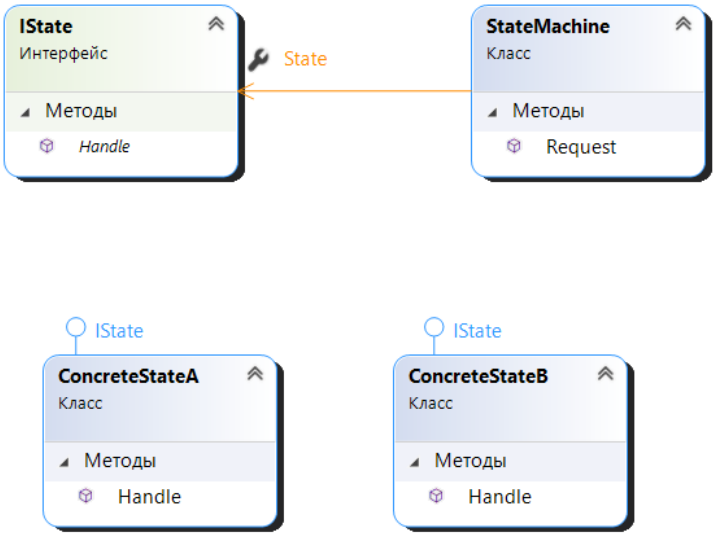

Рисунок 1 – Диаграмма классов

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.

```
using System;

namespace PatternStateLib.AbstractDemo
{
    // Представляет состояния
    interface IState
    {
        void Handle(StateMachine sm);
    }
    // Представляет конкретные состояния A
    class ConcreteStateA : IState
    {
        public void Handle(StateMachine sm)
        {
            Console.Write("Объект {0} в сост. A", sm.ToString());
        }
    }
    // Представляет конкретные состояния B
    class 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();
        }
    }
}
```

Пример. Реализация шаблона Состояние в приложении на языке C#.

Требуется разработать приложение, в котором моделируется работа конечного автомата, в качестве которого выступает навесной замок. Примем, что объект «навесной замок» может находиться в следующих состояниях: Открыт, Закрыт, Неисправен.

Основными операциями объекта «навесной замок» являются:

Закрыть – переводит объект из состояния «Закрыт» в состояние «Открыт»;

Открыть – переводит объект из состояния «Закрыт» в состояние «Открыт»;

Ремонтировать – переводит объект из состояния «Неисправен» в состояние «Открыт».

Переход объекта в состояние «Неисправен» может происходить при вызове операций Открыть и Закрыть случайным образом с вероятностью *p*.

Диаграмма конечных автоматов указанного объекта показана на рис. 2.

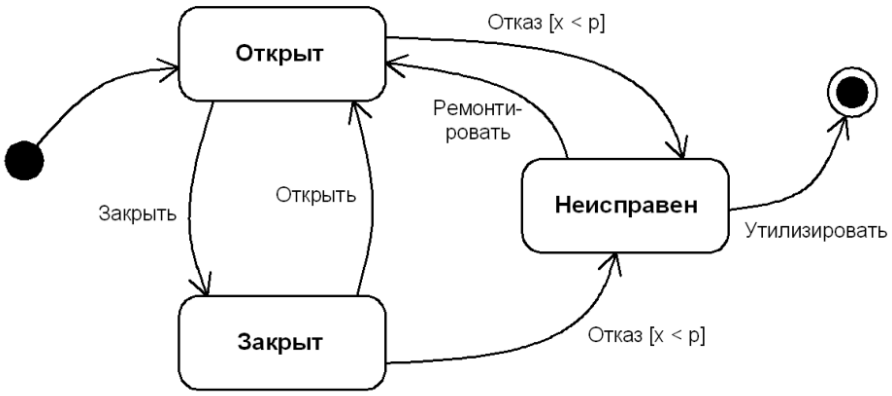

Рисунок 2 – Диаграмма состояний замка

Возможные реакции объекта на выполнение операций в различных состояниях представлены в табл. 1.

Операции	Состояния		
	Открыт	**Закрыт**	**Неисправен**
Закрыть	Переход в со стояние «За крыт»	Исключение «Замок уже за крыт»	Исключение «Замок неис правен»
Открыть	Исключение «Замок уже от крыт»	Переход в со стояние «От крыт»	Исключение «Замок неис правен»
Ремонтировать	Исключение «Замок не тре бует ремонта»	Исключение «Замок не тре бует ремонта»	Переход в со стояние «От крыт»

Исходный код класса **Padlock**

```
using System;

namespace PatternStateLib
{
    public class Padlock
    {
        public string Serial { get; set; }
        public double Unfixprob { get; set; }
        public IState State { get; set; }

        Random rnd;

        public Padlock(string serial, double prob)
        {
            Serial = serial;
            Unfixprob = prob;
            rnd = new Random();
            State = new UnlockedState();
        }

        public void ModelUnfix()
        {
            if (rnd.NextDouble() < Unfixprob)
            {
                State = new UnfixedState();
                throw new Exception(string.Format("Замок {0} сломался!", Serial));
            }
        }

        public void Lock()
        {
            ModelUnfix();
            State.Lock(this);
        }
        public void Unlock()
        {
            ModelUnfix();
            State.Unlock(this);
        }
        public void Fix()
        {
            State.Fix(this);
```

```
        }

        public override string ToString()
        {
            return string.Format("Данные о замке: \n" +
                "Серийный номер: {0}\n" +
                "Вероятность поломки: {1}\n" +
                "Текущее состояние: {2}\n",
                Serial, Unfixprob, State.Note);
        }
    }
}
```

Исходный код интерфейса **IState**

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PatternStateLib
{
    public interface IState
    {
        string Note { get; }
        void Lock(Padlock padlock);
        void Unlock(Padlock padlock);
        void Fix(Padlock padlock);
    }
}
```

Исходный код класса **LockedState**

```
using System;

namespace PatternStateLib
{
    public class LockedState : IState
    {
        string note;

        public LockedState()
        {
            note = "Замок закрыт";
        }

        public string Note => note;

        public void Fix(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} не нуждается в ремонте!",
padlock.Serial));
        }

        public void Lock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} уже закрыт!", padlock.Serial));
        }
```

```
        public void Unlock(Padlock padlock)
        {
            padlock.State = new UnlockedState();
        }
    }
}
```

Исходный код класса **UnlockedState**

```
using System;

namespace PatternStateLib
{
    public class UnlockedState : IState
    {
        string note;

        public string Note => note;

        public UnlockedState()
        {
            note = "Открыт";
        }

        public void Fix(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} не нуждается в ремонте!",
padlock.Serial));
        }

        public void Lock(Padlock padlock)
        {
            padlock.State = new LockedState();
        }

        public void Unlock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} уже открыт!", padlock.Serial));
        }
    }
}
```

Исходный код класса **UnfixedState**

```
using System;

namespace PatternStateLib
{
    public class UnfixedState : IState
    {
        string note;

        public UnfixedState()
        {
            note = "Неисправен";
        }
```

```
        public string Note => note;

        public void Fix(Padlock padlock)
        {
            padlock.State = new UnlockedState();
        }

        public void Lock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} неисправен!", padlock.Serial));
        }

        public void Unlock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} неисправен!", padlock.Serial));
        }
    }
}
```

Демо-приложение для разработанной библиотеки выполнено в виде приложения Windows Forms. Внешний вид формы приложения приведен на рисунке 3;

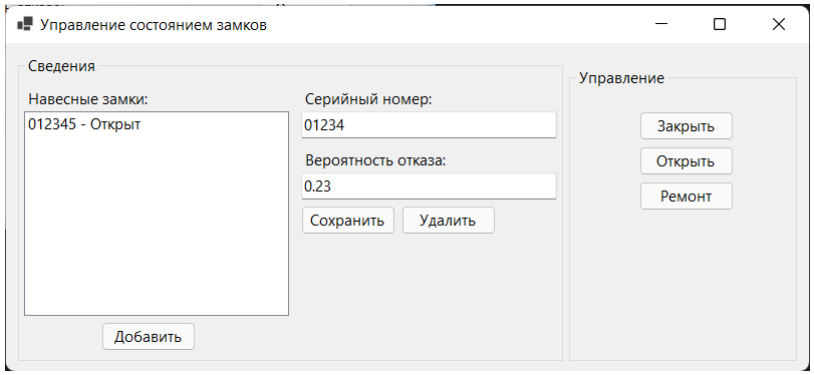

Рисунок 3 – Внешний вид формы

Исходный код класса формы

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using PatternStateLib;

namespace StateFormsApp
{
    public partial class Form1 : Form
    {
        List<Padlock> padlocks;

        public Form1()
        {
```

```
    InitializeComponent();
    padlocks = new List<Padlock>();
    padlocks.Add(new Padlock("012345", 0.23));
}

private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Управление состоянием замков";
    textBox1.Text = "01234";
    textBox2.Text = "0.23";
    GenderPadlocksList();
}

private void GenderPadlocksList()
{
    listBox1.Items.Clear();
    foreach (Padlock p in padlocks)
    {
        listBox1.Items.Add(string.Format("{0} - {1}",
            p.Serial, p.State.Note));
    }
}

private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    int index = listBox1.SelectedIndex;
    if (index < 0) return;

    textBox1.Text = padlocks[index].Serial;
    textBox2.Text = padlocks[index].Unfixprob.ToString();
}

private void button4_Click(object sender, EventArgs e)
{
    int index = listBox1.SelectedIndex;
    if (index < 0) return;
    try
    {
        padlocks[index].Lock();
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message);
    }
    GenderPadlocksList();
}

private void button5_Click(object sender, EventArgs e)
{
    int index = listBox1.SelectedIndex;
    if (index < 0) return;
    try
    {
        padlocks[index].Unlock();
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message);
    }
    GenderPadlocksList();
}

private void button6_Click(object sender, EventArgs e)
{
    int index = listBox1.SelectedIndex;
```

```
            if (index < 0) return;
            try
            {
                padlocks[index].Fix();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
            GenderPadlocksList();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double p;
            if (!double.TryParse(textBox2.Text, out p))
            {
                return;
            }
            padlocks.Add(new Padlock(textBox1.Text, p));
            GenderPadlocksList();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            int index = listBox1.SelectedIndex;
            if (index < 0) return;
            double p;
            if (!double.TryParse(textBox2.Text, out p))
            {
                return;
            }
            padlocks[index].Serial = textBox1.Text;
            padlocks[index].Unfixprob = p;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            int index = listBox1.SelectedIndex;
            if (index < 0) return;
            padlocks.RemoveAt(index);
            GenderPadlocksList();
        }
    }
}
```

Задание на лабораторную работу

Разработать иерархию классов с использованием шаблона Состояние согласно вариантам.

Варианты заданий для разработки приложения с использованием шаблона Состояние

№ вар.	Классы, их атрибуты и операции	Состояния
1, 7, 13, 19	**Телефон.** ***Атрибуты***: номер, баланс, вероятность поступления звонка. ***Операции***: Позвонить, Ответить на звонок, Завершить разговор, Пополнить баланс.	Ожидание, Звонок, Разговор, Заблокирован (баланс отрицательный)
2, 8, 14, 20	**Банкомат.** ***Атрибуты***: ID, общая сумма денег в банкомате, вероятность отсутствия связи с банком. ***Операции***: Ввести PIN-код, Снять заданную сумму, Завершить работу, Загрузить деньги в банкомат.	Ожидание, Аутентификация пользователя, Выполнение операций, Заблокирован (денег нет)
3, 9, 15, 21	**Грузовой лифт.** ***Атрибуты***: текущий этаж, грузоподъёмность, вероятность отключения электроэнергии. ***Операции***: Вызвать на заданный этаж, Загрузить, Разгрузить, Восстановить подачу энергии.	Покой, Движение, Перегружен, Нет питания
4, 10, 16, 22	**Пулемёт.** ***Атрибуты***: скорострельность, число патронов в магазине, вероятность осечки. ***Операции***: Нажать курок, Отпустить курок, Перезарядить, Сменить ствол.	Готовность, Стрельба, Перегрев, Отсутствие патронов
5, 11, 17, 23	**Смеситель.** ***Атрибуты***: максимальный напор холодной (горячей) воды, вероятность отключения воды. ***Операции***: Повернуть кран с холодной (горячей) водой на заданный угол, Переключить воду на излив, переключить воду на лейку, Восстановить подачу воды	Закрыт, Выливание воды через излив (носик), Выливание воды через душевую лейку, Нет воды.
6, 12, 18, 24	**Принтер.** ***Атрибуты*:** модель, число листов в лотке, количество краски в картридже, вероятность зажатия бумаги.	Ожидание, Печать документа, Зажатие бумаги, Отказ (отсутствует бумага или краска)

	Операции: Печатать, Загрузить бумагу, Извлечь зажатую бумагу, Заправить картридж.	