**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет**

По лабораторной работе № 3

На тему: «Шаблоны поведения»

По курсу: «Архитектура, технологии и ИС разработки ПО»

Вариант № 10

Выполнили:

студенты гр. КТбо3-6

Гришак В. С.

Проверил:

доцент кафедры МОП ЭВМ

Лутай В. Н.

Таганрог 2022

Цель работы: использовать необходимый паттерн проектирования при написании программы.

1. Вариант задания 10

Создайте Команду для управления включением-выключением света в доме, кондиционера, отпиранием и запиранием дверей.

1. Спецификация классов с комментариями

Были разработаны классы Door, Lamp, Conditioner. Класс Conditioner содержит свойство IsTurnOn (флаг состояния работы) и два метода: Freeze (морозить), TurnOff (выключиться). Класс Door содержит свойство IsOpen (флаг состояния открытости) и два метода: Open (открыть), Close (закрыть). Класс Lamp содержит свойство IsSining (флаг включённого состояния) и два метода Shine (светить), TurnOff (выключиться).

Для их управления был создан интерфейс ICommend с двумя методами Execute (выполнение команды) и Undo (отмена команды).

Были разработаны 6 типов команд:

ConditionerOnCommand – включить кондиционер,

ConditionerOffCommand – выключить кондиционер,

DoorOpenCommand – открыть дверь,

DoorCloseCommand – закрыть дверь,

LampsOnCommand – включить освещение,

LampsOffCommand – выключить освещение.

Для организации доступа к данным командам был разработан класс Pult.

Для демонстрации работы программы был разработан класс ConsoleInteractor.

1. Используемые математические зависимости и алгоритмы, например, вычисления дня недели по дате или алгоритма пересечения фигур.

Никакие математические зависимости не были применены.

1. Диаграмма классов предоставлена на рисунке 1.

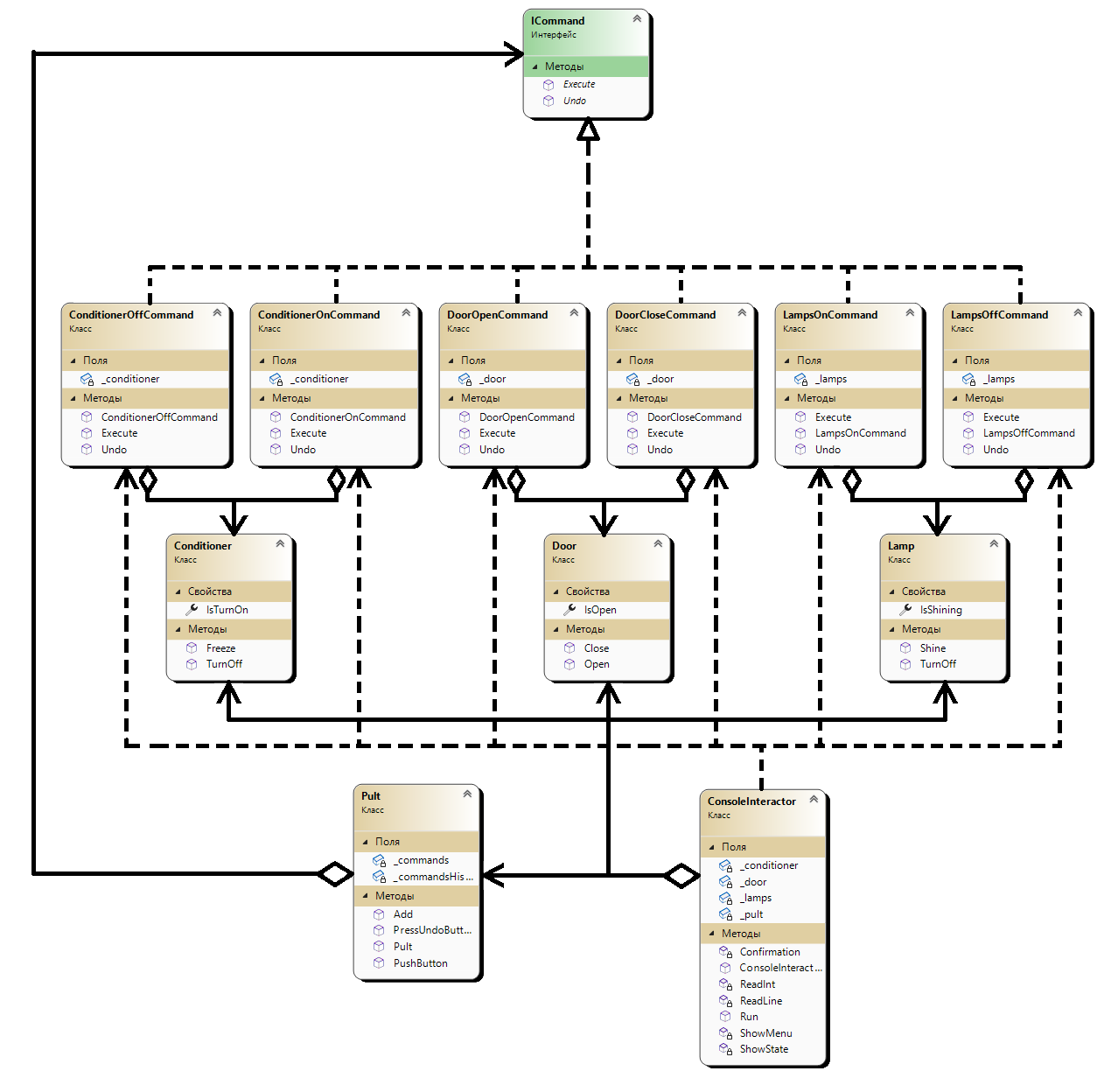


Рисунок 1 – Диаграмма классов

Вывод: в ходе выполнения лабораторной работы был успешно применён паттерн проектирования Команда.

**ПРИЛОЖЕНИЕ**

ConsoleInteractor.cs:

using patterns\_laba\_3.Commands;

using patterns\_laba\_3.Invoker;

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3;

public class ConsoleInteractor

{

private readonly IList<Lamp> \_lamps;

private readonly Door \_door;

private readonly Conditioner \_conditioner;

private readonly Pult \_pult;

public ConsoleInteractor()

{

\_lamps = new List<Lamp>()

{

new Lamp(),

new Lamp(),

new Lamp(),

new Lamp(),

new Lamp(),

};

\_door = new Door();

\_conditioner = new Conditioner();

\_pult = new Pult();

\_pult.Add(PultsCommands.ConditionerOn, new ConditionerOnCommand(\_conditioner));

\_pult.Add(PultsCommands.ConditionerOff, new ConditionerOffCommand(\_conditioner));

\_pult.Add(PultsCommands.DoorOpen, new DoorOpenCommand(\_door));

\_pult.Add(PultsCommands.DoorClose, new DoorCloseCommand(\_door));

\_pult.Add(PultsCommands.LampsOn, new LampsOnCommand(\_lamps));

\_pult.Add(PultsCommands.LampsOff, new LampsOffCommand(\_lamps));

}

private void ShowState()

{

Console.WriteLine($"Дверь {(\_door.IsOpen ? "открыта" : "закрыта")}");

Console.WriteLine($"Сплит система {(\_conditioner.IsTurnOn ? "включена" : "выключена")}");

if (\_lamps.All((lamp) => lamp.IsShining))

{

Console.WriteLine("Освещение дома включено");

}

else if (\_lamps.All((lamp) => !lamp.IsShining))

{

Console.WriteLine("Освещение дома выключено");

}

else

{

Console.WriteLine("Освещение дома частично включена");

}

}

private void ShowMenu()

{

Console.WriteLine($"{0} - Выйти");

Console.WriteLine($"{1} - Показать меню");

Console.WriteLine($"{2} - Отменить последнюю команду");

Console.WriteLine($"{3} - Показать состояние дома");

Console.WriteLine($"{4} - Включить кондиционер");

Console.WriteLine($"{5} - Выключить кондиционер");

Console.WriteLine($"{6} - Открыть дверь");

Console.WriteLine($"{7} - Закрыть дверь");

Console.WriteLine($"{8} - Включить освещение в доме");

Console.WriteLine($"{9} - Выключить освещение в доме");

}

private static string? ReadLine()

{

Console.Write(">> ");

return Console.ReadLine();

}

private static int ReadInt()

{

string? s = ReadLine();

if (int.TryParse(s, out int code))

{

return code;

}

else

return -1;

}

private static bool Confirmation()

{

Console.WriteLine("Для подтверждения нажмите 1");

if (Console.ReadKey(true).Key == ConsoleKey.D1)

{

return true;

}

else

{

Console.WriteLine("Команда отменена");

return false;

}

}

public void Run()

{

try

{

var seporator = new string('=', 20);

int operation;

ShowMenu();

do {

operation = ReadInt();

switch (operation)

{

case 0:

if (!Confirmation())

{

operation = 0;

}

break;

case 1:

ShowMenu();

break;

case 2:

if (Confirmation())

\_pult.PressUndoButton();

break;

case 3:

ShowState();

break;

case 4:

\_pult.PushButton(PultsCommands.ConditionerOn);

break;

case 5:

\_pult.PushButton(PultsCommands.ConditionerOff);

break;

case 6:

\_pult.PushButton(PultsCommands.DoorOpen);

break;

case 7:

\_pult.PushButton(PultsCommands.DoorClose);

break;

case 8:

\_pult.PushButton(PultsCommands.LampsOn);

break;

case 9:

\_pult.PushButton(PultsCommands.LampsOff);

break;

default:

Console.WriteLine("Такой команды нет!!!");

break;

}

Console.WriteLine();

} while (operation != 0);

}

catch (Exception e)

{

Console.WriteLine("Что-то пошло не так");

Console.WriteLine(e.ToString());

}

}

}

Program.cs:

using patterns\_laba\_3;

new ConsoleInteractor().Run();

ConditionerOffCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class ConditionerOffCommand : ICommand

{

private readonly Conditioner \_conditioner;

public ConditionerOffCommand(Conditioner conditioner)

{

\_conditioner = conditioner;

}

public void Execute()

{

if (\_conditioner.IsTurnOn)

{

\_conditioner.TurnOff();

}

else

{

Console.WriteLine("Сплит система и так выключена");

}

}

public void Undo()

{

new ConditionerOnCommand(\_conditioner).Execute();

}

}

ConditionerOnCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class ConditionerOnCommand : ICommand

{

private readonly Conditioner \_conditioner;

public ConditionerOnCommand(Conditioner conditioner)

{

\_conditioner = conditioner;

}

public void Execute()

{

if (\_conditioner.IsTurnOn)

{

Console.WriteLine("Сплит система и так работает!");

}

else

{

\_conditioner.Freeze();

}

}

public void Undo()

{

new ConditionerOffCommand(\_conditioner).Execute();

}

}

DoorCloseCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class DoorCloseCommand : ICommand

{

private readonly Door \_door;

public DoorCloseCommand(Door door)

{

\_door = door;

}

public void Execute()

{

if (\_door.IsOpen)

{

\_door.Close();

}

else

{

Console.WriteLine("Дверь и так закрыта");

}

}

public void Undo()

{

new DoorOpenCommand(\_door).Execute();

}

}

DoorOpenCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class DoorOpenCommand : ICommand

{

private readonly Door \_door;

public DoorOpenCommand(Door door)

{

\_door = door;

}

public void Execute()

{

if (\_door.IsOpen)

{

Console.WriteLine("Дверь и так открыта!");

}

else

{

\_door.Open();

}

}

public void Undo()

{

new DoorCloseCommand(\_door).Execute();

}

}

ICommand.cs:

namespace patterns\_laba\_3.Commands;

public interface ICommand

{

void Execute();

void Undo();

}

LampsOffCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class LampsOffCommand : ICommand

{

private readonly IList<Lamp> \_lamps;

public LampsOffCommand(IList<Lamp> lamps)

{

\_lamps = lamps;

}

public void Execute()

{

if (\_lamps.All((lamp) => !lamp.IsShining))

{

Console.WriteLine("Свет и так выключен");

}

else

{

foreach (var lamp in \_lamps)

{

if (lamp.IsShining)

{

lamp.TurnOff();

}

else

{

Console.WriteLine("Лампа и так выключена");

}

}

}

}

public void Undo()

{

new LampsOnCommand(\_lamps).Execute();

}

}

LampsOnCommand.cs:

using patterns\_laba\_3.Receivers;

namespace patterns\_laba\_3.Commands;

public class LampsOnCommand : ICommand

{

private readonly IList<Lamp> \_lamps;

public LampsOnCommand(IList<Lamp> lamps)

{

\_lamps = lamps;

}

public void Execute()

{

if (\_lamps.All((lamp) => lamp.IsShining))

{

Console.WriteLine("Свет и так включен");

}

else

{

foreach (var lamp in \_lamps)

{

if (lamp.IsShining)

{

Console.WriteLine("Лампа и так включена");

}

else

{

lamp.Shine();

}

}

}

}

public void Undo()

{

new LampsOffCommand(\_lamps).Execute();

}

}

Pult.cs:

using patterns\_laba\_3.Commands;

namespace patterns\_laba\_3.Invoker;

public class Pult

{

private readonly IDictionary<PultsCommands, ICommand> \_commands;

private readonly Stack<ICommand> \_commandsHistory;

public Pult()

{

\_commands = new Dictionary<PultsCommands, ICommand>();

\_commandsHistory = new Stack<ICommand>();

}

public void Add(PultsCommands pultsCommand, ICommand command)

{

\_commands.Add(pultsCommand, command);

}

public void PushButton(PultsCommands pultsCommand)

{

\_commands[pultsCommand].Execute();

// добавляем выполненную команду в историю команд

\_commandsHistory.Push(\_commands[pultsCommand]);

}

public void PressUndoButton()

{

if (\_commandsHistory.Count > 0)

{

ICommand undoCommand = \_commandsHistory.Pop();

undoCommand.Undo();

}

}

}

PultsCommands.cs:

namespace patterns\_laba\_3.Invoker;

public enum PultsCommands

{

ConditionerOn,

ConditionerOff,

DoorOpen,

DoorClose,

LampsOn,

LampsOff

}

Conditioner.cs:

namespace patterns\_laba\_3.Receivers;

public class Conditioner

{

public bool IsTurnOn { get; private set; } = false;

public void Freeze()

{

IsTurnOn = true;

Console.WriteLine("Сплит Система включается");

Thread.Sleep(500);

Console.WriteLine("Сплит Система включена");

Console.WriteLine("Повеяло прохладой");

}

public void TurnOff()

{

IsTurnOn = false;

Console.WriteLine("Сплит Система выключается");

Thread.Sleep(500);

Console.WriteLine("Сплит Система выключена");

}

}

Door.cs:

namespace patterns\_laba\_3.Receivers;

public class Door

{

public bool IsOpen { get; private set; } = false;

public void Open()

{

IsOpen = true;

Console.WriteLine("Дверь открывается");

Thread.Sleep(500);

Console.WriteLine("Дверь открыта");

}

public void Close()

{

IsOpen = false;

Console.WriteLine("Дверь закрывается");

Thread.Sleep(500);

Console.WriteLine("Дверь закрыта");

}

}

Lamp.cs:

namespace patterns\_laba\_3.Receivers;

public class Lamp

{

public bool IsShining { get; private set; } = false;

public void Shine()

{

IsShining = true;

Console.WriteLine("Лампочка зажглась");

}

public void TurnOff()

{

IsShining = false;

Console.WriteLine("Лампочка выключилась");

}

}