**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 5

**З дисципліни:** *“Моделювання та аналіз програмного забезпечення”*

**На тему:** *“* *Структурні шаблони”*

**Лектор:**

доц. каф. ПЗ

Сердюк П.В.

**Виконав:**

ст. гр. ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

асист. каф. ПЗ

Цимбалюк Т.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_\_ .

Львів – 2022

**Тема роботи:** Структурні шаблони.

**Мета роботи:** Ознайомлення з структурними шаблонами на основі мови програмування C#.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. Крім того, що Фасад дозволяє знизити загальну складність програми, він також допомагає винести код, який залежить від зовнішньої системи, в одне місце.

Декоратор — це структурний патерн, який дозволяє динамічно додавати нові поведінки об’єктам, розміщаючи їх в об’єктах-обгортках.

Замісник — це структурний патерн, що створює об’єкт, який виступає прошарком між клієнтом та реальним сервісним об’єктом. Замісник отримує виклики від клієнта, виконує свою функцію (контроль доступу, кешування, зміна запиту та інше), а потім передає виклик сервісному об’єктові.

Легковаговик — це структурний патерн проектування, що дає змогу вмістити більшу кількість об’єктів у відведеній оперативній пам’яті. Легковаговик заощаджує пам’ять, розподіляючи спільний стан об’єктів між собою, замість зберігання однакових даних у кожному об’єкті.

Компонувальник — це структурний патерн проектування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт.

**ЗАВДАННЯ**

Розробити структурні шаблони проектування відповідно до індивідуального завдання.

Обов’язкові для реалізації

1. Фасад.

Реалізувати 2 шаблони на вибір:

1. Декоратор.

2. Компонувальник.

3. Замісник.

4. Легковаговик.

Індивідуальне завдання: Створення казино.

Обрані шаблони: Декоратор та Замісник.

**ХІД ВИКОНАННЯ**

1. Реалізація шаблону Фасад. Створений клас GameFacade для полегшення роботи з системою казино. Відповідає за створення гри та найм робітників. Також добавлений метод для підрахунку прибутку за день.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public class GameFacade

{

public Casino Casino;

public IEmployer croupierEmployee = new CroupierEmployer();

public IEmployer guardEmployee = new GuardEmployer();

public SwindlerBuilder swindlerBuilder = new SwindlerBuilder();

public SwindlerDirector swindlerDirector = new SwindlerDirector();

private State state = new StateEasy();

private IStrategy \_complexity = new StrategyEasy();

public void SetStrategy(IStrategy strategy)

{

\_complexity = strategy;

string mes = "";

if(\_complexity is StrategyEasy)

{

mes = TransitionTo(new StateEasy());

}

else if (\_complexity is StrategyHard)

{

mes = TransitionTo(new StateHard());

}

Form1.GetInstance().CallMessage($"You change game complexity!\n{mes}\n{state.GetMessage()}", "Info");

}

public string TransitionTo(State state)

{

this.state = state;

return "You change game state!";

}

public Swindler BuildSwindler(int type)

{

return swindlerDirector.BuildSwindler(swindlerBuilder, type);

}

public void HireNewCroupier(int index)

{

Casino.tables[index].AddCroupier(Hire(croupierEmployee));

}

public void HireNewGuard(int index)

{

Casino.tables[index].AddGuard(Hire(guardEmployee));

}

private IEmployee Hire(IEmployer employer)

{

return employer.Hire(Casino.random);

}

public void NewGame()

{

Casino.NewGame();

Casino = Casino.GetInstance();

}

public void PassDay()

{

Casino.PassDay(\_complexity.ProfitRatio(), \_complexity.SwindlerProbability());

}

private static GameFacade instance;

public static GameFacade GetInstance()

{

if (instance == null)

instance = new GameFacade();

return instance;

}

}

}

1. Реалізація шаблону Замісник. Створено клас ProxyPlayer, який може бути або звичайним гравцем або шулером в залежності від різних умов.

interface IPlayer

{

int Level { set; get; }

int Profit();

}

public class Swindler : IPlayer

{

public Swindler()

{

Level = 0;

Experience = 0;

CheatInBigCasinos = false;

HasTeam = false;

KnowMathematics = false;

WasInOurCasino = false;

}

public Swindler(int level)

{

Level = level;

}

public int Profit()

{

int max = Casino.GetInstance().Balance;

if(max <= 20000)

{

max = 20000;

}

int sum = Level \* 300;

return Math.Min(sum, max);

}

public void CalculateLevel()

{

Level = 1 \* (Experience + 1);

if (CheatInBigCasinos)

{

Level \*= 3;

}

if (HasTeam)

{

Level \*= 2;

}

if (KnowMathematics)

{

Level += 3;

}

if (WasInOurCasino)

{

Level += 2;

}

}

public int Level { set; get; }

public int Experience { set; get; }

public bool CheatInBigCasinos { set; get; }

public bool HasTeam { set; get; }

public bool KnowMathematics { set; get; }

public bool WasInOurCasino { set; get; }

}

public class ProxyPlayer : IPlayer

{

public Swindler \_swindler = null;

public int Level { set; get; } = Casino.random.Next(0, 100);

public int Profit()

{

if(\_swindler == null)

{

return Level;

}

else

{

return \_swindler.Profit();

}

}

public int GetLevel()

{

if (\_swindler == null)

{

return Level;

}

else

{

return \_swindler.Level;

}

}

public bool IsSwindler()

{

return \_swindler != null;

}

public void Clear()

{

\_swindler = null;

Level = Casino.random.Next(0, 100);

}

public void SetSwindler(Swindler swindler)

{

\_swindler = swindler;

}

}

1. Реалізація шаблону Декоратор. Створено інтерфейс ISwindlerInfo для наслідування нащадками і реалізації базового методу декоратора. Також створено абстрактний клас SwindlerMessageDecorator для подальшого наслідування з різними варіаціями виведення повідомлення про шулера.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public abstract class ISwindlerInfo

{

public abstract string GetInfo(Swindler swindler, int tableIndex);

}

public class SwindlerInfo : ISwindlerInfo

{

public override string GetInfo(Swindler swindler, int tableIndex)

{

return $"To {tableIndex + 1} table come swindler ";

}

}

public abstract class SwindlerMessageDecorator : ISwindlerInfo

{

protected ISwindlerInfo swindlerInfo;

public void SetPlayer(ISwindlerInfo info)

{

swindlerInfo = info;

}

public override string GetInfo(Swindler swindler, int tableIndex)

{

if (swindlerInfo != null)

{

return swindlerInfo.GetInfo(swindler, tableIndex);

}

else return string.Empty;

}

}

public class DefaultSwindlerMessageDecorator : SwindlerMessageDecorator

{

public override string GetInfo(Swindler swindler, int tableIndex)

{

return base.GetInfo(swindler, tableIndex);

}

}

public class LevelSwindlerMessageDecorator : SwindlerMessageDecorator

{

public override string GetInfo(Swindler swindler, int tableIndex)

{

var str = base.GetInfo(swindler, tableIndex);

str += $"with lvl {swindler.Level}";

return str;

}

}

public class AllInfoSwindlerMessageDecorator : SwindlerMessageDecorator

{

public override string GetInfo(Swindler swindler, int tableIndex)

{

var str = base.GetInfo(swindler, tableIndex);

str += $"with lvl {swindler.Level}.\n";

str += $"He has {swindler.Experience} year of experience.\n";

if (swindler.KnowMathematics)

{

str += $"He knows mathematic.\n";

}

if (swindler.HasTeam)

{

str += $"He has team.\n";

}

if (swindler.CheatInBigCasinos)

{

str += $"He has already cheated in big casinos.\n";

}

if (swindler.WasInOurCasino)

{

str += $"And he has already been in our casino.\n";

}

return str;

}

}

public class DecoratorHelper

{

public static string Message(int type, Swindler player, int tableIndex)

{

SwindlerMessageDecorator decorator = new DefaultSwindlerMessageDecorator();

switch (type)

{

case 1:

decorator = new LevelSwindlerMessageDecorator();

break;

case 2:

decorator = new AllInfoSwindlerMessageDecorator();

break;

}

decorator.SetPlayer(new SwindlerInfo());

return decorator.GetInfo(player, tableIndex);

}

}

}

**РЕЗУЛЬТАТИ**

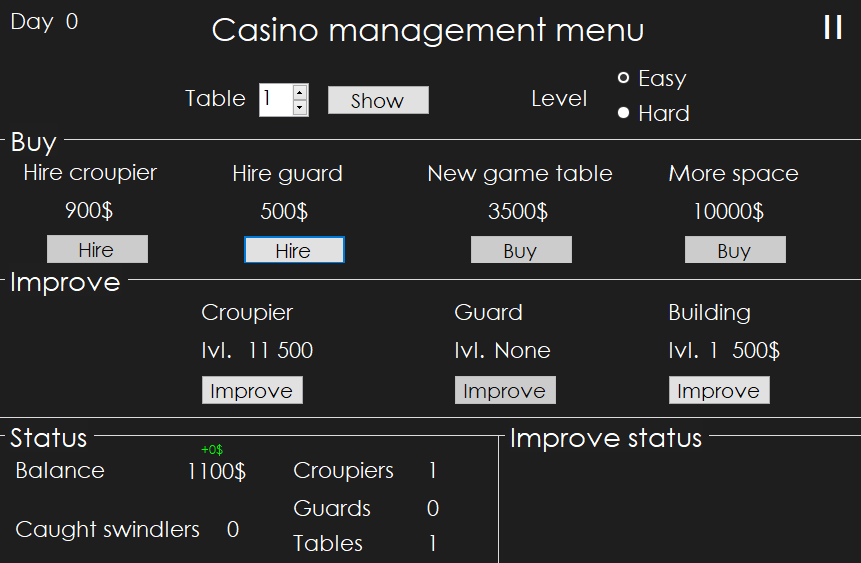
****

Рис. 1. Меню управління казино

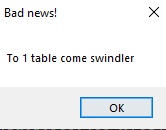
****

Рис. 2. Спливаюче вікно про прихід шулера (шаблон Декоратор)

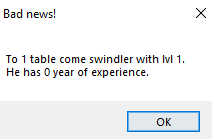
****

Рис. 3. Інше повідомлення у спливаючому вікні про прихід шулера (шаблон Декоратор)

****

Рис. 4. Замісник є шулером тому показується шулер



Рис. 5. Замісник є гравцем

**ВИСНОВКИ**

На цій лабораторній роботі я додав до інтерфейсу гри вікно із ігровим столом і візуалізацією дилера, шулера та охоронця. Також навчився працювати та використав шаблони Фасад, Замісник та Декоратор.

Фасад використовується, як організатор для складної системи класів та дозволяє зручно викликати методи цих класів не вдаючись в подробиці реалізації. Цей шаблон в моєму проекті використовується для створення гри, найму охоронця та дилера, та для виклику основної функції класу Casino.

Декоратор використовується для різного виводу статистики про конкретного шулера, якщо він прийшов. Спочатку реалізовувався звичайний декоратор, що сповіщає просто про прихід шулера до певного столу, однак далі створені були додаткові нащадки базового класу для більш детального опису шуллера, що прийшов.

Замісник я використав для зміни учасника столу з гравця на шулера і навпаки. З певною ймовірністю учасник може стати шулером і тоді виграє більше грошей, аніж звичайний гравець.