**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 6

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

**На тему:** *“Поведінкові 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:** Поведінкові шаблони.

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

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Стратегія — це поведінковий патерн, який виносить набір алгоритмів у власні класи і робить їх взаємозамінними. Інші об’єкти містять посилання на об’єкт-стратегію та делегують їй роботу. Програма може підмінити цей об’єкт іншим, якщо потрібен інший спосіб вирішення завдання.

Стан — це поведінковий патерн, що дозволяє динамічно змінювати поведінку об’єкта при зміні його стану. Поведінки, які залежать від стану, переїзджають в окремі класи. Початковий клас зберігає посилання на один з таких об’єктів-станів та делегує йому роботу.

Спостерігач — це поведінковий патерн, який дозволяє об’єктам повідомляти інші об’єкти про зміни свого стану. При цьому спостерігачі можуть вільно підписуватися і відписуватись від цих повідомлень.

Шаблонний метод — це поведінковий патерн, який визначає кістяк алгоритму в суперкласі та змушує підкласи реалізувати конкретні кроки цього алгоритму.

Команда — це поведінковий патерн, що дозволяє загортати запити або прості операції в окремі об’єкти. Це дозволяє відкладати виконання команд, вибудовувати їх у чергу, а також зберігати історію і робити скасування.

Ланцюжок обов’язків — це поведінковий патерн, що дозволяє передавати запит ланцюжком потенційних обробників до тих пір, поки один з них не обробить його.

Посередник — це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.

**ЗАВДАННЯ**

Розробити поведінкові шаблони проектування відповідно до індивідуального завдання.

Шаблони для реалізації (на вибір)

1. Ланцюжок відповідальностей (Chain of Responsibility)

2. Стан (State)

3. Стратегія (Strategy)

4. Шаблонний метод (Template Method)

5. Шаблон Команда (Command).

6. Посередник (Mediator),

7. Спостерігач (Observer)

Індивідуальне завдання: Створення казино.

Обрані шаблони: Стратегія, Спостерігач, Стан.

**ХІД ВИКОНАННЯ**

1. Реалізація шаблону Стратегія. Реалізовано два класи для різної складності гри : легкої та важкої.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public interface IStrategy

{

float ProfitRatio();

int SwindlerProbability();

}

public class StrategyEasy : IStrategy

{

public float ProfitRatio()

{

return 0.3F;

}

public int SwindlerProbability()

{

return 10;

}

}

public class StrategyHard : IStrategy

{

public float ProfitRatio()

{

return 1;

}

public int SwindlerProbability()

{

return 30;

}

}

}

1. Реалізація шаблону Стан. Створено класи для сповіщення про зміну рівня складності.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public abstract class State

{

public abstract string GetMessage();

}

public class StateEasy : State

{

public override string GetMessage()

{

return "Now game level is Easy!";

}

}

public class StateHard : State

{

public override string GetMessage()

{

return "Now game level is Hard!";

}

}

}

1. Реалізація шаблону Спостерігач. Створено класи, що реагують на початок гри та прихід нового дня.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public interface IObserver

{

void Update(ISubject subject);

}

public interface ISubject

{

void Attach(IObserver observer);

void Detach(IObserver observer);

void Notify();

}

public class DayObserver : IObserver

{

private List<string> advices = new List<string>{

"Track your spending to improve your finances",

"Create a realistic monthly budget",

"Build up your savings—even if it takes time",

"Pay your bills on time every month",

"Cut back on recurring charges",

"Save up cash to afford big purchases",

"Start an investment strategy",

"Make sure you’re paying the best prices",

"Save up for big purchases",

"Contribute to savings regularly",

"Start investing",

"Contribute to your retirement",

"Plan for large expenses",

"Build an emergency fund",

"Take a look at your income" };

public void Update(ISubject subject)

{

if ((subject as Casino).Day == 0)

{

return;

}

if ((subject as Casino).Day % 2 == 0)

{

Form1.GetInstance().CallMessage($"Today is your {(subject as Casino).Day} day as casino manager. Some advice:\n{advices[Casino.random.Next(0, advices.Count)]}", "Dayly news!");

}

else if((subject as Casino).Day % 2 == 1)

{

Form1.GetInstance().CallMessage($"Today is your {(subject as Casino).Day} day as casino manager.", "Dayly news!");

}

}

}

public class NewGameObserver : IObserver

{

public void Update(ISubject subject)

{

if ((subject as Casino).tables.Count == 0 && (subject as Casino).Day == 0)

{

Form1.GetInstance().CallMessage("You start the new game!", "Welcome!");

}

}

}

}

Використання створених шаблонів у основних класах:

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;

}

}

}

public class Casino : ISubject

{

private Casino()

{

Attach(new NewGameObserver());

Attach(new DayObserver());

Notify();

Profit = 0;

Balance = 5000;

PreviousBalance = Balance;

CaughtSwindlers = 0;

}

public int Profit { set; get; }

public int Balance { set; get; }

public int PreviousBalance { set; get; }

public int CaughtSwindlers { set; get; }

public int Day { set; get; }

private List<IObserver> \_observers = new List<IObserver>();

public void Attach(IObserver observer)

{

\_observers.Add(observer);

}

public void Detach(IObserver observer)

{

\_observers.Remove(observer);

}

public void Notify()

{

foreach (var observer in \_observers)

{

observer.Update(this);

}

}

}

**РЕЗУЛЬТАТИ**

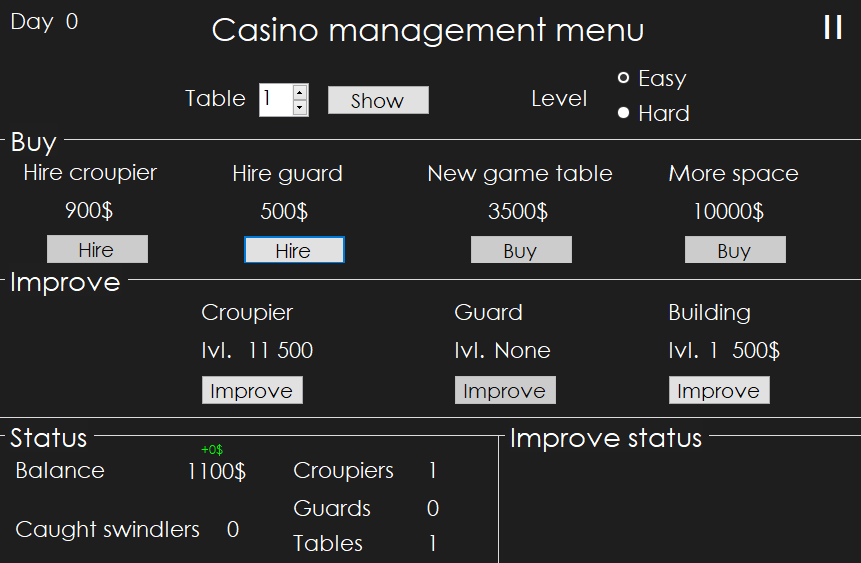
****

Рис. 1. Меню управління казино

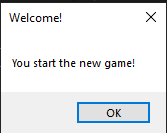
****

Рис. 2. Спливаюче вікно про старт гри (шаблон Спостерігач)

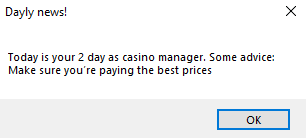
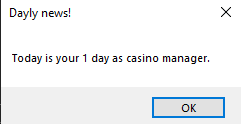
****

Рис. 3. Спливаюче вікно про новий день (шаблон Спостерігач)

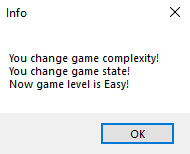
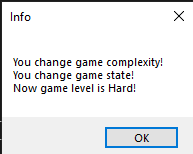


Рис. 4. Спливаюче вікно про зміну рівня складності (шаблон Стан та Стратегія)

**ВИСНОВКИ**

На даній лабораторній роботі я ознайомився із поведінковими шаблонами проектування, зокрема використав на практиці такі шаблони: стратегія, спостерігач та стан.

Стратегію я використав для змінення рівня складності гри з легкого на важкий та навпаки. Для даного шаблону створюються декілька стратегії, що наслідують одного батька та перевантажують основні методи, як їм потрібно. Далі ж користувач просто обирає потрібну стратегію і вже з нею працює. Такий шаблон зручно використовувати, коли користувач може сам вибирати різні стратегії розвитку або складності.

Наступний шаблон – спостерігач, логіка його в тому, що самі функції цього шаблону мають можливість сповіщати інші об’єкти про своє виконання, подібно реалізовано до звичайної підписки на певну послугу, тобто в моєму проекті клас Casino сповіщає всіх своїх підписників, що він змінився і якщо підходить умова, вони викликають свої методи. Шаблон використав для сповіщення про створення гри та для подальшого сповіщення про кожен новий день.

Стан я використав для перевірки чи змінювалась стратегія. Цей шаблон дуже схожий до шаблону Стратегія, однак є певні відмінності. Суть цього методу, полягає в створені створенні класу, який вміє міняти свій стан і що дає змогу змінювати свою поведінку, для програми об’єкт весь час виглядає одним, проте по суті в ньому підмінюється керування іншим наперед розписаним класом-станом.