**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 4

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

**На тему:** *“* *Твірні 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:** Твірні шаблони.

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

**TЕОРЕТИЧНІ ВІДОМОСТІ**

  Абстрактна фабрика — це породжуючий патерн проектування, який вирішує проблему створення цілих сімейств пов’язаних продуктів, без прив’язки коду до конкретних класів продуктів. Абстрактна фабрика задає інтерфейс створення всіх доступних типів продуктів, а кожна конкретна реалізація фабрики породжує продукти однієї з варіацій. Клієнтський код викликає методи фабрики для отримання продуктів, замість самостійного створювання їх за допомогою оператора *new*. Фабрика сама стежить за реалізацією.

Одинак — це породжуючий патерн, який гарантує існування тільки одного об’єкта певного класу, а також дозволяє дістатися цього об’єкта з будь-якого місця програми.. Ви не зможете просто взяти і використовувати клас, залежний від одинака, в іншій програмі. Для цього доведеться емулювати там присутність одинака. Найчастіше ця проблема проявляється при написанні юніт-тестів.

Прототип — це породжувальний патерн проектування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації. Патерн Прототип доручає процес копіювання самим об’єктам, які треба скопіювати. Він вводить загальний інтерфейс для всіх об’єктів, що підтримують клонування. Це дозволяє копіювати об’єкти, не прив’язуючись до їхніх конкретних класів.

Будівельник — це породжуючий патерн проектування, який дозволяє створювати об’єкти покроково. На відміну від інших породжуючих патернів, Будівельник дозволяє виготовляти різні продукти, використовуючи один і той же процес будівництва. Патерн можна часто зустріти в C#-коді, особливо там, де необхідним є покрокове створення продуктів або конфігурація складних об’єктів.

**ЗАВДАННЯ**

Розробити твірні шаблони проектування відповідно до індивідуального завдання.

Обов’язкові для реалізації:  
1. Singleton.  
2. Абстрактна фабрика  
Реалізувати один із шаблонів на вибір:  
1. Будівельник.  
2. Прототип.

Індивідуальне завдання: Створення казино.

Обраний шаблон: будівельник.

**ХІД ВИКОНАННЯ**

1. Реалізація патерну Singleton.

Реалізував даний патерн для головного меню та меню паузи, оскільки вони потрібні тільки в одному екземплярі, а також для головного класу Casino.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public sealed class MainMenu

{

private static MainMenu instance;

public static MainMenu GetInstance()

{

if (instance == null)

instance = new MainMenu();

return instance;

}

public void Start()

{

Form1.GetInstance().OpenGames();

}

public void Exit()

{

Form1.GetInstance().Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

class PauseMenu

{

private static PauseMenu instance;

public static PauseMenu GetInstance()

{

if (instance == null)

instance = new PauseMenu();

return instance;

}

public void GoToMainMenu()

{

Form1.GetInstance().OpenMainMenu();

}

public void Resume()

{

Form1.GetInstance().ClosePauseMenu();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public class Casino

{

private Casino()

{

random = new Random();

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; }

public IEmployer croupierEmployee = new CroupierEmployer();

public IEmployer guardEmployee = new GuardEmployer();

public List<IEmployee> Croupiers = new List<IEmployee>();

public List<IEmployee> Guards = new List<IEmployee>();

public SwindlerBuilder swindlerBuilder = new SwindlerBuilder();

public SwindlerDirector swindlerDirector = new SwindlerDirector();

public Building Building = new Building();

public void PassDay()

{

int randomnumber = random.Next(0, 100);

bool swindlerCome = false;

if (randomnumber > 90)

{

swindlerCome = true;

}

int income = Building.Profit();

int cheated = 0;

foreach(var croupier in Croupiers)

{

income += croupier.Income();

}

if (swindlerCome)

{

randomnumber = random.Next(0, (Building.Tables + 1) \* 10);

int count = 0;

for (var i = 1; i < Building.Tables + 1; ++i)

{

if (randomnumber - (Building.Tables + 1 - i) \* 10 < 0)

{

count = i-1;

}

}

var guards = new List<IEmployee>(Guards);

for (var counter = 0; counter < count; ++counter)

{

int[] weights1 = { 36, 62, 77, 89, 96, 99, 100 };

int type = 0;

randomnumber = random.Next(0, 100);

for (var i = 0; i < weights1.Length; ++i)

{

if (randomnumber - weights1[i] < 0)

{

type = i + 1;

break;

}

}

var sw = swindlerDirector.BuildSwindler(swindlerBuilder, type);

int guardsLevel = 0;

bool Robbed = true;

foreach (var g in guards)

{

if(guardsLevel > sw.Level \* 2)

{

Robbed = false;

CaughtSwindlers += 1;

break;

}

if (g.Level < sw.Level \* 2)

{

guards.Remove(g);

guardsLevel += g.Level;

}

else

{

Robbed = false;

CaughtSwindlers += 1;

guards.Remove(g);

break;

}

}

if (Robbed) { cheated += sw.Robbed(); }

}

}

if(cheated > 0)

{

Form1.GetInstance().CallMessage("The swindlers take out of your casino " + cheated.ToString() + "$ today.");

}

Profit = income - (PreviousBalance - Balance) - cheated;

var prof = income - cheated;

Balance += prof;

PreviousBalance = Balance;

Day += 1;

}

public void HireNewCroupier()

{

Croupiers.Add(croupierEmployee.Hire(random));

}

public void HireNewGuard()

{

Guards.Add(guardEmployee.Hire(random));

}

private static Casino instance;

private static Random random;

public static Casino GetInstance()

{

if (instance == null)

instance = new Casino();

return instance;

}

public static void NewGame()

{

instance = new Casino();

}

}

}

2. Реалізація патерну Builder.

Реалізував даний патерн для будування різних шулерів

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public class SwindlerBuilder

{

private Swindler swindler = new Swindler();

public void SetExperience(int experience)

{

swindler.Experience = experience;

}

public void SetCheatInBigCasinos()

{

swindler.CheatInBigCasinos = true;

}

public void SethasTeam()

{

swindler.HasTeam = true;

}

public void SetKnowMathematics()

{

swindler.KnowMathematics = true;

}

public void SetWasInOurCasino()

{

swindler.WasInOurCasino = true;

}

public Swindler ReturnSwindler()

{

swindler.CalculateLevel();

Swindler result = swindler;

swindler = new Swindler();

return result;

}

public void ResetSwindler()

{

swindler = new Swindler();

}

}

public class SwindlerDirector

{

public Swindler Newcommer(SwindlerBuilder builder)

{

builder.ResetSwindler();

return builder.ReturnSwindler();

}

public Swindler Junior(SwindlerBuilder builder)

{

builder.ResetSwindler();

builder.SetExperience(1);

return builder.ReturnSwindler();

}

public Swindler Middle(SwindlerBuilder builder)

{

builder.ResetSwindler();

builder.SetExperience(2);

builder.SetKnowMathematics();

return builder.ReturnSwindler();

}

public Swindler Senior(SwindlerBuilder builder)

{

builder.ResetSwindler();

builder.SetExperience(4);

builder.SetWasInOurCasino();

builder.SetKnowMathematics();

return builder.ReturnSwindler();

}

public Swindler Principle(SwindlerBuilder builder)

{

builder.ResetSwindler();

builder.SetExperience(6);

builder.SetWasInOurCasino();

builder.SetKnowMathematics();

builder.SetCheatInBigCasinos();

return builder.ReturnSwindler();

}

public Swindler Master(SwindlerBuilder builder)

{

builder.ResetSwindler();

builder.SetExperience(10);

builder.SetWasInOurCasino();

builder.SetKnowMathematics();

builder.SetCheatInBigCasinos();

builder.SethasTeam();

return builder.ReturnSwindler();

}

public Swindler MasterRandom(SwindlerBuilder builder)

{

builder.ResetSwindler();

Random random = new Random();

builder.SetExperience(random.Next(10, 40));

builder.SetWasInOurCasino();

builder.SetKnowMathematics();

builder.SetCheatInBigCasinos();

builder.SethasTeam();

return builder.ReturnSwindler();

}

public Swindler BuildSwindler(SwindlerBuilder builder, int type)

{

switch (type)

{

case 1:

return Newcommer(builder);

case 2:

return Junior(builder);

case 3:

return Middle(builder);

case 4:

return Senior(builder);

case 5:

return Principle(builder);

case 6:

return Master(builder);

case 7:

return MasterRandom(builder);

default:

return Newcommer(builder);

}

}

}

public class Swindler

{

public Swindler()

{

Level = 0;

Experience = 0;

CheatInBigCasinos = false;

HasTeam = false;

KnowMathematics = false;

WasInOurCasino = false;

}

public Swindler(int level)

{

Level = level;

}

public int Robbed()

{

int max = Casino.GetInstance().Balance;

if(max <= 20000)

{

max = 20000;

}

int sum = Level \* 300;

return Level;

}

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; }

}

}

3. Реалізація патерну Abstract factory.

Реалізував даний патерн для створення різних працівників з подібними функціями, але різною імплементацією.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public interface IEmployee

{

int GetWage();

void Improve();

int Income();

int ImprovePrice();

int Level { set; get; }

}

public interface IEmployer

{

IEmployee Hire(Random random);

}

public class CroupierEmployer : IEmployer {

public IEmployee Hire(Random random)

{

int randomNumber = random.Next(1, 1000000);

if (randomNumber == 555555)

{

randomNumber = 30;

}

else if (randomNumber < 900000 && randomNumber > 850000)

{

randomNumber = random.Next(8, 15);

}

else

{

randomNumber = random.Next(1, 3);

}

return new Croupier(randomNumber);

}

}

public class GuardEmployer : IEmployer {

public IEmployee Hire(Random random)

{

int randomNumber = random.Next(1, 1000000);

if (randomNumber == 555555)

{

randomNumber = 10;

}

else if (randomNumber < 900000 && randomNumber > 850000)

{

randomNumber = random.Next(5, 8);

}

else

{

randomNumber = random.Next(1, 3);

}

return new Guard(randomNumber);

}

}

public class Croupier : IEmployee

{

public Croupier()

{

Level = 1;

}

public Croupier(int level)

{

Level = level;

}

public int Level { set; get; }

public void Improve()

{

Casino.GetInstance().Balance -= ImprovePrice();

Level += 1;

}

public int ImprovePrice()

{

return 100 + (Level - 1) \* 40;

}

public int GetWage()

{

return 700 + (Level - 1) \* 60;

}

public int Income()

{

return 5 + (Level - 1);

}

}

public class Guard : IEmployee

{

public Guard()

{

Level = 1;

}

public Guard(int level)

{

Level = level;

}

public int Level { set; get; }

public void Improve()

{

Casino.GetInstance().Balance -= ImprovePrice();

Level += 1;

}

public int ImprovePrice()

{

return 70 + (Level - 1) \* 30;

}

public int GetWage()

{

return 700 + (Level - 1) \* 50;

}

public int Income()

{

return 0;

}

}

}

Клас Casino, який використовує дані патерни:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MAPZ\_lab4

{

public class Casino

{

private Casino()

{

random = new Random();

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; }

public IEmployer croupierEmployee = new CroupierEmployer();

public IEmployer guardEmployee = new GuardEmployer();

public List<IEmployee> Croupiers = new List<IEmployee>();

public List<IEmployee> Guards = new List<IEmployee>();

public SwindlerBuilder swindlerBuilder = new SwindlerBuilder();

public SwindlerDirector swindlerDirector = new SwindlerDirector();

public Building Building = new Building();

public void PassDay()

{

int randomnumber = random.Next(0, 100);

bool swindlerCome = false;

if (randomnumber > 90)

{

swindlerCome = true;

}

int income = Building.Profit();

int cheated = 0;

int caught = 0;

foreach (var croupier in Croupiers)

{

income += croupier.Income();

}

if (swindlerCome)

{

randomnumber = random.Next(0, (Building.Tables + 1) \* 10);

int count = 0;

for (var i = 1; i < Building.Tables + 1; ++i)

{

if (randomnumber - (Building.Tables + 1 - i) \* 10 < 0)

{

count = i-1;

}

}

var guards = new List<IEmployee>(Guards);

for (var counter = 0; counter < count; ++counter)

{

int[] weights1 = { 36, 62, 77, 89, 96, 99, 100 };

int type = 0;

randomnumber = random.Next(0, 100);

for (var i = 0; i < weights1.Length; ++i)

{

if (randomnumber - weights1[i] < 0)

{

type = i + 1;

break;

}

}

var sw = swindlerDirector.BuildSwindler(swindlerBuilder, type);

int guardsLevel = 0;

bool Robbed = true;

foreach (var g in guards)

{

if(guardsLevel > sw.Level \* 2)

{

Robbed = false;

CaughtSwindlers += 1;

++caught;

break;

}

if (g.Level < sw.Level \* 2)

{

guards.Remove(g);

guardsLevel += g.Level;

}

else

{

Robbed = false;

CaughtSwindlers += 1;

++caught;

guards.Remove(g);

break;

}

}

if (Robbed) { cheated += sw.Robbed(); }

}

}

if(cheated > 0)

{

Form1.GetInstance().CallMessage("The swindlers take out of your casino " + cheated.ToString() + "$ today.");

}

if (caught > 0)

{

if (caught == 1)

Form1.GetInstance().CallMessage("Your guards caught " + caught.ToString() + " swindler today.");

else

Form1.GetInstance().CallMessage("Your guards caught " + caught.ToString() + " swindlers today.");

}

Profit = income - (PreviousBalance - Balance) - cheated;

var prof = income - cheated;

Balance += prof;

Form1.GetInstance().CallMessage("The swindlers take out of your casino " + 1000.ToString() + "$ today.");

PreviousBalance = Balance;

Day += 1;

}

public void HireNewCroupier()

{

Croupiers.Add(Hire(croupierEmployee));

}

public void HireNewGuard()

{

Guards.Add(Hire(guardEmployee));

}

private IEmployee Hire(IEmployer employer)

{

return employer.Hire(random);

}

private static Casino instance;

private static Random random;

public static Casino GetInstance()

{

if (instance == null)

instance = new Casino();

return instance;

}

public static void NewGame()

{

instance = new Casino();

}

}

}

**РЕЗУЛЬТАТИ**

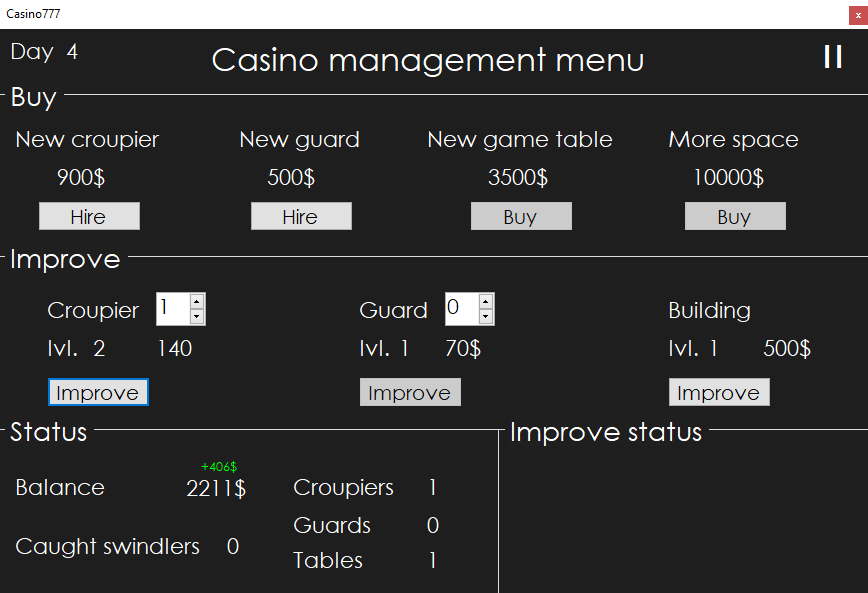
****

Рис. 2. Меню управління казино

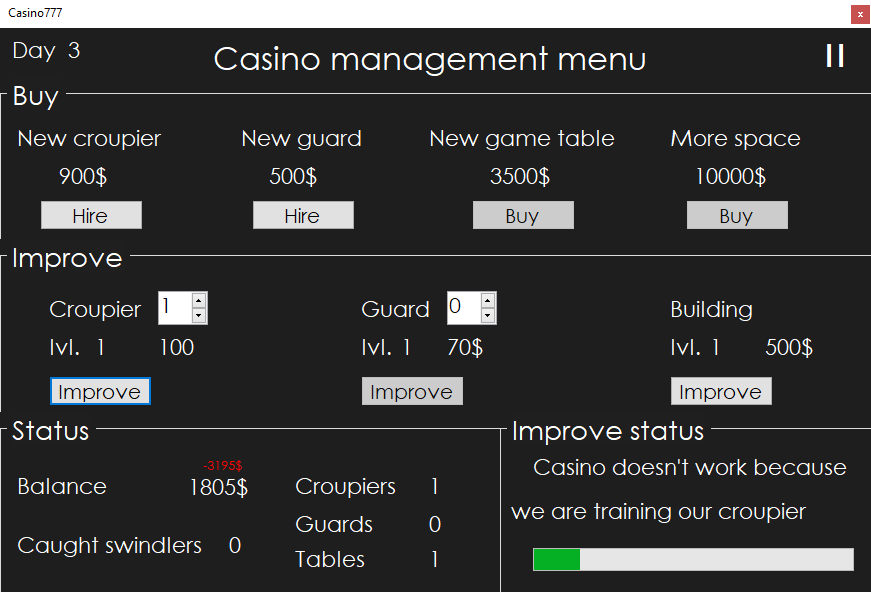
****

Рис. 3. Меню покращень із аналогом Progress Bar

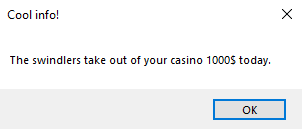
****

Рис. 4. Спливаюче вікно про шулера і кількість втрачених грошей

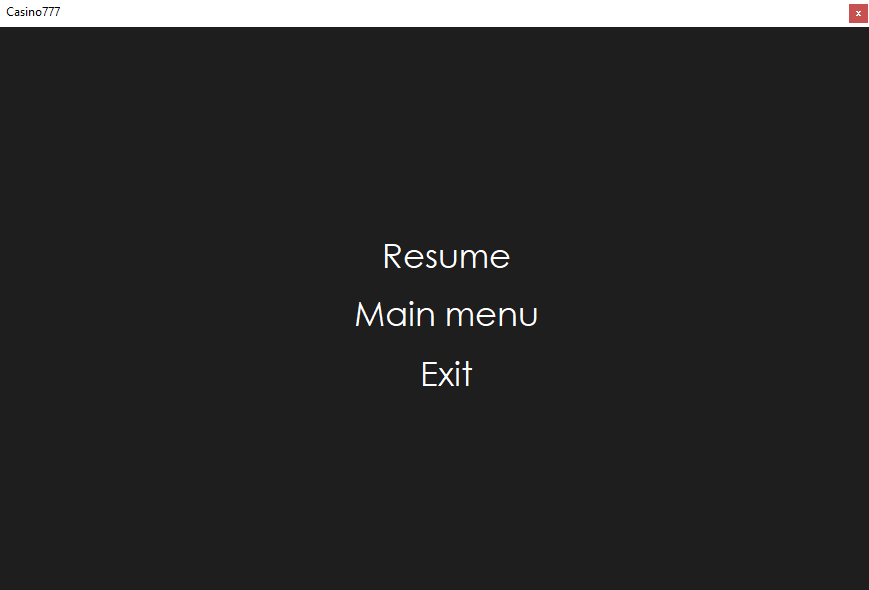
****

Рис. 5. Меню паузи

****

Рис. 6. Головне меню

**ВИСНОВКИ**

На даній лабораторній роботі було ознайомлено з твірними шаблонами Abstract Factory, Singleton, Builder та Prototype. На практиці розроблено і використано Abstract Factory, Singleton та Builder.

Singleton використовується у тих випадках коли потрібне звернення до одного об’єкту класу в межах іменного простору з інших структур. Через особливість об’єктно орієнтованих мов програмування такий клас може бути лише в одному екземплярі для уникання помилок, звідси і назва одинак. На даній лабораторній роботі Singleton використовувався для створення об’єктів класів PauseMenu, MainMenu та Casino. Одинак було використано для того, щоб у кожного елемента був доступ до головного меню та меню паузи, у випадку з Казино для того, щоб ми працювали тільки з одним екземпляром з будь-якого місця коду і не створили помилково новий екземпляр.

Builder використовується для створення складних об’єктів покроково. На даній лабораторній роботі Builder використовувався для створення шуллерів різних типів, що вибиралися випадково з певною вагою. Тобто за допомогою класу SwindleDirector можна створити шулера-новачка або шулера-мастера, також можна самостійно конструювати шулера як заманеться за допомогою методів класу SwindleBuilder.

Abstract factory використовується, коли є потреба в постійному створені подібних, проте різних в імплементації, об’єктів. Абстрактна фабрика надає інтерфейс, що задає всі доступні типи і слідкує, щоб створився потрібний тип продукту. На даній лабораторній роботі Abstract factory використовується для класів Guard та Croupier, оскільки це два наймані працівники і функціонал вони мають однаковий, проте імплементацію різну. Тому клієнтський код (клас Casino) працює з створеними для даного шаблону для цього інтерфейсами.