МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

кафедра програмних засобів

Звіт

з лабораторної роботи № 5

з дисципліни «Проектний практикум»

на тему:

«Практичне використання шаблонів проектування»

Виконав

Ст. гр. КНТ-137 В.В. Козлов

Прийняв

Доцент, к.т.н. Н.О. Миронова

1. Мета роботи

Отримати практичні навички використання шаблонів проектування при розробці програмного продукту.

1. Завдання на лабораторну роботу

Використовуючи знання, отримані при виконанні лабораторних робіт 1-4, вирішити приладну задачу.

* 1. Задача 1: Модель «Годинник на ратуші»

У місті працюють «Ремісники», що використовують для організації часу у своїй роботі «Годинник» на міській ратуші.

«Пекар» о 4 ранку розпалює піч, о 5 замішує тісто, о 6 ставить тісто випікатись, о 7 витягає випічку з печі, о 8 відкриває пекарню для продажу випічки, о 16 зачиняє пекарню. «Чоботяр» о 10 відкриває майстерню, об 11 випиває гальбу пива, о 15 починає співати пісню, о 18 зачиняє майстерню і йде до шинку. «Шинкар» о 16 починає варити пиво, о 17 відчиняє шинок, о 23 зачиняє шинок.

Усі «ремісники» прислухаються до того, яка година, і відповідно до цього вирішують, що їм слід робити. «Годинник» дозволяє чути усім, хто його слухає, котра година, тобто виступає спостережуваним суб’єктом, а ремісники – спостерігачами

Використати шаблон проектування «Одинак» (Singleton) для опису класу «Годинник», шаблон проектування «Спостерігач» (Observer) для опису взаємодії нащадків класу «Ремісник» (observer) з класом «Годинник» (observable) та шаблон проектування «Ітератор» (Iterator) для організації послідовного звертання до списку спостерігачів.

* 1. Задача 2: Модель «Святий Миколай»

Єдиним джерелом подарунків чемним дітям та покарання неслухняним є «Святий Миколай», який приймає у дітей замовлення на подарунки та дарує їм їх, або карає за погані вчинки. У цьому «Святому Миколаю» допомагає «Фабрика добрих подарунків» та «Фабрика поганих подарунків», кожна з яких створює один їстівний та один неїстівний подарунок (методи абстрактної фабрики та абстрактні продукти відповідно).

«Фабрика добрих подарунків» створює солодощі та іграшки, а «Фабрика поганих подарунків» створює гіркі пілюлі та різки.

Для того, щоб отримати подарунок, діти надсилають повідомлення у вигляді двох чисел: кількості хороших і поганих вчинків, які вони скоїли за останній рік.

«Миколай» збирає усі повідомлення від дітей та вирішує, надіслати дитині добрий чи поганий подарунок в залежності від того, яких вчинків більше (добрих чи поганих) – призначає кожній дитині у списку відповідну конкретну фабрику.

У день Святого Миколая повинні спрацювати призначені їм фабрики, а «Миколай» надсилає дітям створені для них подарунки.

Використати шаблон проектування «Одинак» (Singleton) для опису класу «Святий Миколай», шаблон проектування «Абстрактна фабрика» (Abstract Factory) для опису створення подарунків (abstract/concrete product) на фабриках (abstract/concrete factory) та шаблон проектування «Ітератор» (Iterator) для організації послідовного звертання до списку дітей та призначених їм фабрик.

1. Результати виконання роботи
   1. Реалізація першого завдання
      1. Лістинг файлу Task1.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab5.Tasks

{

public interface IPublisher

{

public void Subscribe(ISubscriber subscriber);

public void UnSubscribe(ISubscriber subscriber);

public void NotifySubscribers();

}

public interface ISubscriber

{

public void Update(int time);

}

public abstract class Craftsman : ISubscriber

{

public string Name { get; set; }

public Craftsman(string name)

{

Name = name;

}

public abstract void Update(int time);

}

public class Baker : Craftsman

{

public Baker(string name) : base(name) { ; }

public override void Update(int time)

{

switch(time)

{

case 4:

{

KindleOven();

break;

}

case 5:

{

KneadDough();

break;

}

case 6:

{

PutDoughToOven();

break;

}

case 7:

{

TakeDoughFromOven();

break;

}

case 8:

{

OpenBakery();

break;

}

case 16:

{

CloseBakery();

break;

}

}

}

public void KindleOven()

{

Console.WriteLine($"{Name} kindle the oven");

}

public void KneadDough()

{

Console.WriteLine($"{Name} knead the dough");

}

public void PutDoughToOven()

{

Console.WriteLine($"{Name} put dough to oven");

}

public void TakeDoughFromOven()

{

Console.WriteLine($"{Name} take out the baking from oven");

}

public void OpenBakery()

{

Console.WriteLine($"{Name} open bakery for selling baking");

}

public void CloseBakery()

{

Console.WriteLine($"{Name} close bakery");

}

}

public class Shoemaker : Craftsman

{

public Shoemaker(string name) : base(name) { ; }

public override void Update(int time)

{

switch(time)

{

case 10:

{

OpenWorkshop();

break;

}

case 11:

{

DrinkBeer();

break;

}

case 15:

{

StartSing();

break;

}

case 18:

{

CloseWorkshop();

break;

}

}

}

public void OpenWorkshop()

{

Console.WriteLine($"{Name} open workshop");

}

public void DrinkBeer()

{

Console.WriteLine($"{Name} drink a beer can");

}

public void StartSing()

{

Console.WriteLine($"{Name} start singing a song");

}

public void CloseWorkshop()

{

Console.WriteLine($"{Name} close workshop and goes to the ham");

}

}

public class Barkeeper : Craftsman

{

public Barkeeper(string name) : base(name) { ; }

public override void Update(int time)

{

switch(time)

{

case 16:

{

BrewBeer();

break;

}

case 17:

{

OpenHam();

break;

}

case 23:

{

CloseHam();

break;

}

}

}

public void BrewBeer()

{

Console.WriteLine($"{Name} start brewing beer");

}

public void OpenHam()

{

Console.WriteLine($"{Name} open the ham");

}

public void CloseHam()

{

Console.WriteLine($"{Name} close the ham");

}

}

public interface IIterator

{

public bool IsEnd();

public ISubscriber GetNext();

public void Update();

}

public interface ICollection

{

public void Add(ISubscriber subscriber);

public void Remove(ISubscriber subscriber);

public ISubscriber this[int index] { get; set; }

public int Size();

public IIterator CreateIterator();

}

public class CraftsmanList : ICollection

{

private List<ISubscriber> \_Subscribers { get; set; }

public CraftsmanList()

{

\_Subscribers = new List<ISubscriber>();

}

public void Add(ISubscriber subscriber)

{

if (!\_Subscribers.Contains(subscriber))

\_Subscribers.Add(subscriber);

}

public void Remove(ISubscriber subscriber)

{

if (\_Subscribers.Contains(subscriber))

\_Subscribers.Remove(subscriber);

}

public ISubscriber this[int index]

{

get

{

return \_Subscribers[index];

}

set

{

\_Subscribers[index] = value;

}

}

public int Size()

{

return \_Subscribers.Count;

}

public IIterator CreateIterator()

{

return new Iterator(this);

}

}

public class Iterator : IIterator

{

private ICollection \_Collection { get; set; }

private int \_IterationState { get; set; }

public Iterator(ICollection collection)

{

\_Collection = collection;

Update();

}

public bool IsEnd()

{

if (\_IterationState >= \_Collection.Size())

return true;

else

return false;

}

public ISubscriber GetNext()

{

if (!IsEnd())

return \_Collection[\_IterationState++];

else

throw new Exception("Iterator reach the end...");

}

public void Update()

{

\_IterationState = 0;

}

}

public class Clock : IPublisher

{

private static Clock \_Clock { get; set; }

private CraftsmanList \_Craftsmans { get; set; }

public int Time { get; set; }

private Clock()

{

\_Craftsmans = new CraftsmanList();

Time = 0;

}

public static Clock GetInstance()

{

if (\_Clock == null)

\_Clock = new Clock();

return \_Clock;

}

public void NextHour()

{

Time = Time == 23 ? 0 : ++Time;

Console.WriteLine($"Current time: {Time}:00");

NotifySubscribers();

}

public void SimulateAllDay()

{

Time = 23;

Console.WriteLine("All day simulation: ");

for (int i = 0; i < 24; ++i)

NextHour();

}

public void Subscribe(ISubscriber subscriber)

{

\_Craftsmans.Add(subscriber);

}

public void UnSubscribe(ISubscriber subscriber)

{

\_Craftsmans.Remove(subscriber);

}

public void NotifySubscribers()

{

IIterator Iterator = \_Craftsmans.CreateIterator();

while(!Iterator.IsEnd())

{

Iterator.GetNext().Update(Time);

}

}

}

}

* 1. Реалізація другого завдання
     1. Лістинг файлу Task2.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab5.Tasks

{

public interface IEdibleGift

{

public string ShowEdibleGift();

}

public interface IInedibleGift

{

public string ShowInedibleGift();

}

public interface IGiftFactory

{

public IEdibleGift CreateEdibleGift();

public IInedibleGift CreateInedibleGift();

}

public class GoodEdibleGift : IEdibleGift

{

public GoodEdibleGift() { ; }

public string ShowEdibleGift()

{

return "sweets";

}

}

public class BadEdibleGift : IEdibleGift

{

public BadEdibleGift() {; }

public string ShowEdibleGift()

{

return "bitter pills";

}

}

public class GoodInedibleGift : IInedibleGift

{

public GoodInedibleGift() { ; }

public string ShowInedibleGift()

{

return "toys";

}

}

public class BadInedibleGift : IInedibleGift

{

public BadInedibleGift() {; }

public string ShowInedibleGift()

{

return "cutting";

}

}

public class GoodGiftFactory : IGiftFactory

{

public GoodGiftFactory() { ; }

public IEdibleGift CreateEdibleGift()

{

return new GoodEdibleGift();

}

public IInedibleGift CreateInedibleGift()

{

return new GoodInedibleGift();

}

}

public class BadGiftFactory : IGiftFactory

{

public BadGiftFactory() {; }

public IEdibleGift CreateEdibleGift()

{

return new BadEdibleGift();

}

public IInedibleGift CreateInedibleGift()

{

return new BadInedibleGift();

}

}

public class Gift

{

public IEdibleGift EdibleGift { get; set; }

public IInedibleGift InedibleGift { get; set; }

public Gift(IGiftFactory GiftFactory)

{

EdibleGift = GiftFactory.CreateEdibleGift();

InedibleGift = GiftFactory.CreateInedibleGift();

}

public void ShowGift()

{

Console.WriteLine($"{EdibleGift.ShowEdibleGift()} and {InedibleGift.ShowInedibleGift()}");

}

}

public class Child

{

public string Name { get; set; }

public Child(string name)

{

Name = name;

}

public void SendGift(Gift gift)

{

Console.Write($"{Name} received a gift: ");

gift.ShowGift();

}

public void SendLetter(Nicholas SaintNicholas, int good, int bad)

{

SaintNicholas.SendLetter(this, good, bad);

}

}

public interface IChildAndFactoryListIterator

{

public bool IsEnd();

public Tuple<Child, IGiftFactory> GetNext();

public void Update();

}

public interface IChildAndFactoryList

{

public void Add(Child child, IGiftFactory giftFactory);

public void Remove(Child child);

public Tuple<Child, IGiftFactory> this[int index] { get; set; }

public int Size();

public ChildAndFactoryListIterator CreateIterator();

}

public class ChildAndFactoryList : IChildAndFactoryList

{

private List<Child> Child { get; set; }

private List<IGiftFactory> GiftFactory { get; set; }

public ChildAndFactoryList()

{

Child = new List<Child>();

GiftFactory = new List<IGiftFactory>();

}

public void Add(Child child, IGiftFactory giftFactory)

{

if (Child.Contains(child))

{

GiftFactory[Child.IndexOf(child)] = giftFactory;

}

else

{

Child.Add(child);

GiftFactory.Add(giftFactory);

}

}

public void Remove(Child child)

{

if (Child.Contains(child))

{

GiftFactory.RemoveAt(Child.IndexOf(child));

Child.Remove(child);

}

}

public Tuple<Child, IGiftFactory> this[int index]

{

get

{

return new Tuple<Child, IGiftFactory>(Child[index], GiftFactory[index]);

}

set

{

Child[index] = value.Item1;

GiftFactory[index] = value.Item2;

}

}

public int Size()

{

return Child.Count;

}

public ChildAndFactoryListIterator CreateIterator()

{

return new ChildAndFactoryListIterator(this);

}

}

public class ChildAndFactoryListIterator : IChildAndFactoryListIterator

{

private IChildAndFactoryList \_Collection { get; set; }

private int \_IterationState { get; set; }

public ChildAndFactoryListIterator(IChildAndFactoryList collection)

{

\_Collection = collection;

Update();

}

public bool IsEnd()

{

if (\_IterationState >= \_Collection.Size())

return true;

else

return false;

}

public Tuple<Child, IGiftFactory> GetNext()

{

if (!IsEnd())

return \_Collection[\_IterationState++];

else

throw new Exception("Iterator reach the end...");

}

public void Update()

{

\_IterationState = 0;

}

}

public class Nicholas

{

private static Nicholas \_Nicholas { get; set; }

private IGiftFactory \_GoodGiftFactory { get; set; }

private IGiftFactory \_BadGiftFactory { get; set; }

private ChildAndFactoryList \_ChildAndFactoryList { get; set; }

private Nicholas()

{

\_GoodGiftFactory = new GoodGiftFactory();

\_BadGiftFactory = new BadGiftFactory();

\_ChildAndFactoryList = new ChildAndFactoryList();

}

public static Nicholas GetInstance()

{

if (\_Nicholas == null)

{

\_Nicholas = new Nicholas();

return \_Nicholas;

}

else

return \_Nicholas;

}

public void SendLetter(Child child, int good, int bad)

{

if (good > bad)

\_ChildAndFactoryList.Add(child, \_GoodGiftFactory);

else

\_ChildAndFactoryList.Add(child, \_BadGiftFactory);

}

public void CelebrateSaintNicholasDay()

{

Console.WriteLine("Celebrate Saint Nicholas day: ");

IChildAndFactoryListIterator Iterator = \_ChildAndFactoryList.CreateIterator();

while (!Iterator.IsEnd())

{

Tuple<Child, IGiftFactory> ChildAndFactory = Iterator.GetNext();

ChildAndFactory.Item1.SendGift(new Gift(ChildAndFactory.Item2));

}

\_ChildAndFactoryList = new ChildAndFactoryList();

}

}

}

* 1. Виконання завдань
     1. Лістинг файлу Program.cs

using PP\_Lab5.Tasks;

using System;

namespace PP\_Lab5

{

class Program

{

static void Main(string[] args)

{

showTask1();

showTask2();

}

static void showTask1()

{

Console.WriteLine("Task1: ");

Baker ExperiencedBaker = new Baker("Experienced baker");

Shoemaker ExperiencedShoemaker = new Shoemaker("Experienced shoemaker");

Barkeeper ExperiencedBarkeeper = new Barkeeper("Experienced barkeeper");

Clock BigBen = Clock.GetInstance();

BigBen.Subscribe(ExperiencedBaker);

BigBen.Subscribe(ExperiencedShoemaker);

BigBen.Subscribe(ExperiencedBarkeeper);

BigBen.SimulateAllDay();

Console.WriteLine();

}

static void showTask2()

{

Console.WriteLine("Task2: ");

Nicholas SaintNicholas = Nicholas.GetInstance();

Child Valeriy = new Child("Valeriy");

Child Maria = new Child("Maria");

Child Ilon = new Child("Ilon");

Valeriy.SendLetter(SaintNicholas, 10, 1);

Maria.SendLetter(SaintNicholas, 10, 5);

Ilon.SendLetter(SaintNicholas, 1, 10);

SaintNicholas.CelebrateSaintNicholasDay();

Console.WriteLine();

}

}

}

* + 1. Скріншоти роботи програми

На рисунку 3.1 наведено результати виконання програми, яка демонструє виконання усіх завдань лабораторної роботи №5.



Рисунок 3.1 – Резульати виконання програми

Висновки

Протягом виконання цієї лабораторної роботи отримав практичні навички використання шаблонів проектування при розробці програмного продукту.

Використовуючи знання, отримані при виконанні лабораторних робіт 1-4, написав програму мовою програмування C#, яка з використанням шаблонів проектування програмного забезпечення вирішує приладні задачі.