МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

кафедра програмних засобів

Звіт

з лабораторної роботи № 3

з дисципліни «Проектний практикум»

на тему:

«Шаблони поведінки. Частина 1»

Виконав

Ст. гр. КНТ-137 В.В. Козлов

Прийняв

Доцент, к.т.н. Н.О. Миронова

1. Мета роботи

Вивчити шаблони поведінки та отримати базові навички із застосування шаблонів Ітератор (Iterator), Посередник (Mediator), Спостерігач (Observer), Стратегія (Strategy), Ланцюг обов'язків (Chain of Responsibility), Відвідувач (Visitor).

1. Завдання на лабораторну роботу

Вивчити шаблони поведінки, їхню загальну характеристику та призначення. Для кожного з цих шаблонів:

* вивчити призначення шаблону, альтернативні назви, призначення, випадки коли його застосування є доцільним та результати такого застосування;
* знати особливості реалізації шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
* вільно володіти структурою шаблону, призначенням його класів та відносинами між ними.

За допомогою джерел зі списку рекомендованих для виконання лабораторної роботи, виконати програмну реалізацію шаблонів поведінки.

1. Результати виконання роботи
   1. Реалізація паттерну Chain of responsibility
      1. Лістинг файлу PatternChainOfResponsibility.cs

using System;

namespace PP\_Lab3.Patterns

{

public interface IHandler

{

public void setNext(IHandler handler);

public void handle(string request);

}

public abstract class BaseHandler : IHandler

{

private IHandler \_Next { get; set; }

public void setNext(IHandler handler)

{

\_Next = handler;

}

public virtual void handle(string request)

{

if (\_Next != null)

\_Next.handle(request);

}

}

public class ErrorHandler : BaseHandler

{

public override void handle(string request)

{

Console.WriteLine($"Error handler received: {request}");

base.handle(request);

}

}

public class ValidationHandler : BaseHandler

{

public override void handle(string request)

{

Console.WriteLine($"Validation handler received: {request}");

base.handle(request);

}

}

}

* 1. Реалізація паттерну Iterator
     1. Лістинг файлу PatternIterator.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab3.Patterns

{

public interface IIterator

{

public bool isEnd();

public int getNext();

public void setToStart();

}

public interface ICollection

{

public void Add(int value);

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

public int Size();

public IIterator createFrontIterator();

public IIterator createBackIterator();

}

public class Queue : ICollection

{

private List<int> \_Array { get; set; }

public Queue()

{

\_Array = new List<int>();

}

public void Add(int value)

{

\_Array.Add(value);

}

public int this[int index]

{

get

{

return \_Array[index];

}

set

{

\_Array[index] = value;

}

}

public int Size()

{

return \_Array.Count;

}

public IIterator createFrontIterator()

{

return new FrontIterator(this);

}

public IIterator createBackIterator()

{

return new BackIterator(this);

}

}

public class FrontIterator : IIterator

{

private ICollection \_Collection { get; set; }

private int \_IterationState { get; set; }

public FrontIterator(ICollection c)

{

\_Collection = c;

setToStart();

}

public bool isEnd()

{

return \_IterationState >= \_Collection.Size();

}

public int getNext()

{

if (!isEnd())

return \_Collection[\_IterationState++];

else

throw new Exception("Iterator has reached the end");

}

public void setToStart()

{

\_IterationState = 0;

}

}

public class BackIterator : IIterator

{

private ICollection \_Collection { get; set; }

private int \_IterationState { get; set; }

public BackIterator(ICollection c)

{

\_Collection = c;

setToStart();

}

public bool isEnd()

{

return \_IterationState < 0;

}

public int getNext()

{

if (!isEnd())

return \_Collection[\_IterationState--];

else

throw new Exception("Iterator has reached the end");

}

public void setToStart()

{

\_IterationState = \_Collection.Size() - 1;

}

}

}

* 1. Реалізація паттерну Mediator
     1. Лістинг файлу PatternMediator.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab3.Patterns

{

public interface IMediator

{

public void Notify(Component component, string message);

}

public abstract class Component

{

protected IMediator \_Mediator { get; set; }

public Component(IMediator mediator)

{

\_Mediator = mediator;

}

public void Notify(string message)

{

\_Mediator.Notify(this, message);

}

}

public class Plane : Component

{

public string Name { get; set; }

public Plane(string name, IMediator mediator) : base(mediator)

{

Name = name;

}

public void TryToLanding()

{

Notify($"Bort {Name} request landing permission");

}

public void AcceptLanding()

{

Console.WriteLine($"Bort {Name} landed successfully");

}

public void SetInformation(string message)

{

Console.WriteLine("Bort {0} get message : {1}", Name, message);

}

}

public class Airport : Component

{

public string Name { get; set; }

public bool isEmptyRunway { get; set; }

public Airport(string name, IMediator mediator) : base(mediator)

{

Name = name;

isEmptyRunway = true;

}

public void EmptyRunway()

{

isEmptyRunway = true;

Notify("Runway is empty now");

}

public void NotEmptyRunway()

{

isEmptyRunway = false;

Notify("Runway is not empty now");

}

public void SetInformation(string message)

{

Console.WriteLine("Airport {0} get message : {1}", Name, message);

}

}

public class Dispatcher : IMediator

{

public List<Plane> Planes { get; set; }

public Airport DispatcherAirport { get; set; }

public Dispatcher()

{

Planes = new List<Plane>();

}

public void Notify(Component component, string message)

{

if(component == DispatcherAirport)

{

foreach (Plane plane in Planes)

plane.SetInformation(message);

}

else

{

DispatcherAirport.SetInformation(message);

if(DispatcherAirport.isEmptyRunway)

{

((Plane)component).SetInformation("Landing allowed");

((Plane)component).AcceptLanding();

}

else

{

((Plane)component).SetInformation("Landing denied");

}

}

}

}

}

* 1. Реалізація паттерну Observer
     1. Лістинг файлу PatternObserver.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab3.Patterns

{

public interface IPublisher

{

public void Subscribe(ISubscriber subscriber);

public void UnSubscribe(ISubscriber subscriber);

public void NotifySubscribers();

}

public interface ISubscriber

{

public void Update(string message);

}

public class Store : IPublisher

{

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

public Store()

{

\_Subscribers = new List<ISubscriber>();

}

public void Subscribe(ISubscriber subscriber)

{

if(!\_Subscribers.Contains(subscriber))

\_Subscribers.Add(subscriber);

}

public void UnSubscribe(ISubscriber subscriber)

{

if (\_Subscribers.Contains(subscriber))

\_Subscribers.Remove(subscriber);

}

public void NotifySubscribers()

{

foreach (ISubscriber subscriber in \_Subscribers)

subscriber.Update("The store has a brand new IPhone 9!");

}

}

public class Man : ISubscriber

{

public string Name { get; set; }

public Man(string name)

{

Name = name;

}

public void Update(string message)

{

Console.WriteLine($"Man {Name} received message: {message}");

}

}

public class Woman : ISubscriber

{

public string Name { get; set; }

public Woman(string name)

{

Name = name;

}

public void Update(string message)

{

Console.WriteLine($"Woman {Name} received message: {message}");

}

}

}

* 1. Реалізація паттерну Strategy
     1. Лістинг файлу PatternStrategy.cs

using System;

using System.Collections.Generic;

namespace PP\_Lab3.Patterns

{

public interface IStrategy

{

public void Sort<T>(List<T> arr, int size) where T : IComparable;

}

public class BubbleSort : IStrategy

{

public BubbleSort() { ;}

public void Sort<T>(List<T> arr, int size) where T : IComparable

{

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

for(int j = size - 1; j > i; --j)

{

if(arr[j].CompareTo(arr[j - 1]) < 0)

{

T temp = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = temp;

}

}

}

}

public class ReverseBubbleSort : IStrategy

{

public ReverseBubbleSort() {; }

public void Sort<T>(List<T> arr, int size) where T : IComparable

{

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

for (int j = size - 1; j > i; --j)

{

if (arr[j].CompareTo(arr[j - 1]) > 0)

{

T temp = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = temp;

}

}

}

}

public class NewVector<T> where T : IComparable

{

private List<T> Array { get; set; }

private IStrategy Strategy { get; set; }

public NewVector(IStrategy strategy)

{

Array = new List<T>();

Strategy = strategy;

}

public void Sort()

{

Strategy.Sort(Array, Array.Count);

}

public int Size()

{

return Array.Count;

}

public T this[int index]

{

get

{

return Array[index];

}

set

{

Array[index] = value;

}

}

public void Add(T value)

{

Array.Add(value);

}

public void SetStrategy(IStrategy strategy)

{

Strategy = strategy;

}

}

}

* 1. Реалізація паттерну Visitor
     1. Лістинг файлу PatternVisitor.cs

using System;

namespace PP\_Lab3.Patterns

{

public interface IVisitor

{

public void Show(Car car);

public void Show(Airplane airplane);

}

public interface IElement

{

public void accept(IVisitor visitor);

}

public class Car : IElement

{

public string Name { get; set; }

public Car(string name)

{

Name = name;

}

public void accept(IVisitor visitor)

{

visitor.Show(this);

}

}

public class Airplane : IElement

{

public string Name { get; set; }

public Airplane(string name)

{

Name = name;

}

public void accept(IVisitor visitor)

{

visitor.Show(this);

}

}

public class Traveller : IVisitor

{

public Traveller() { ;}

public void Show(Car car)

{

Console.WriteLine("I will ride by car: {0}", car.Name);

}

public void Show(Airplane airplane)

{

Console.WriteLine("I will fly by airplane: {0}", airplane.Name);

}

}

}

* 1. Використання паттернів
     1. Лістинг файлу Program.cs

using System;

using System.Collections.Generic;

using PP\_Lab3.Patterns;

namespace PP\_Lab3

{

class Program

{

static void Main(string[] args)

{

showIterator();

showMediator();

showObserver();

showStretegy();

showChainOfResponsibility();

showVisitor();

}

static void showIterator()

{

Console.WriteLine("Iterator:");

Queue IntArray = new Queue();

for (int i = 1; i <= 5; ++i)

IntArray.Add(i);

IIterator Front = IntArray.createFrontIterator();

IIterator Back = IntArray.createBackIterator();

Console.WriteLine("Front iterator: ");

while (!Front.isEnd())

Console.Write(Front.getNext() + " ");

Console.WriteLine();

Console.WriteLine("Back iterator: ");

while (!Back.isEnd())

Console.Write(Back.getNext() + " ");

Console.WriteLine();

Console.WriteLine();

}

static void showMediator()

{

Console.WriteLine("Mediator:");

Dispatcher AirportDispatcher = new Dispatcher();

Plane V001 = new Plane("V001", AirportDispatcher);

Plane V002 = new Plane("V002", AirportDispatcher);

Airport Heathrow = new Airport("Heathrow", AirportDispatcher);

AirportDispatcher.DispatcherAirport = Heathrow;

AirportDispatcher.Planes.Add(V001);

AirportDispatcher.Planes.Add(V002);

Heathrow.EmptyRunway();

V001.TryToLanding();

Heathrow.NotEmptyRunway();

V002.TryToLanding();

Console.WriteLine();

}

static void showObserver()

{

Console.WriteLine("Observer:");

Man Valeriy = new Man("Valeriy");

Woman Maria = new Woman("Maria");

Man Ilon = new Man("Ilon");

Store AppleStore = new Store();

AppleStore.Subscribe(Valeriy);

AppleStore.Subscribe(Maria);

AppleStore.Subscribe(Ilon);

Console.WriteLine("Notify subscribers: ");

AppleStore.NotifySubscribers();

Console.WriteLine();

}

static void showStretegy()

{

Console.WriteLine("Strategy:");

NewVector<int> Array = new NewVector<int>(new BubbleSort());

for (int i = 5; i > 0; --i)

Array.Add(i);

Console.WriteLine("Before sort: ");

for (int i = 0; i < Array.Size(); ++i)

Console.Write(Array[i] + " ");

Console.WriteLine();

Array.Sort();

Console.WriteLine("After bubble sort: ");

for (int i = 0; i < Array.Size(); ++i)

Console.Write(Array[i] + " ");

Console.WriteLine();

Array.SetStrategy(new ReverseBubbleSort());

Array.Sort();

Console.WriteLine("After reverse bubble sort: ");

for (int i = 0; i < Array.Size(); ++i)

Console.Write(Array[i] + " ");

Console.WriteLine();

Console.WriteLine();

}

static void showChainOfResponsibility()

{

Console.WriteLine("Chain of responsibility:");

IHandler Rule = new ErrorHandler();

Console.WriteLine("One rule handler: ");

Rule.handle("Data");

Rule.setNext(new ValidationHandler());

Console.WriteLine("Two rule handler: ");

Rule.handle("Information");

Console.WriteLine();

}

static void showVisitor()

{

Console.WriteLine("Visitor:");

Traveller Valeriy = new Traveller();

List<IElement> Transport = new List<IElement>();

Transport.Add(new Car("ZAZ 1102 Tavria Nova"));

Transport.Add(new Airplane("AN-225 Mriya"));

Transport.Add(new Car("Ford Fusion"));

Console.WriteLine("Hi, Valeriy. What kind of transport will you travel: ");

foreach (IElement transport in Transport)

transport.accept(Valeriy);

Console.WriteLine();

}

}

}

* + 1. Скріншоти роботи програми

На рисунку 3.1 наведено результати виконання програми, яка демонструє використання шаблонів поведінки.

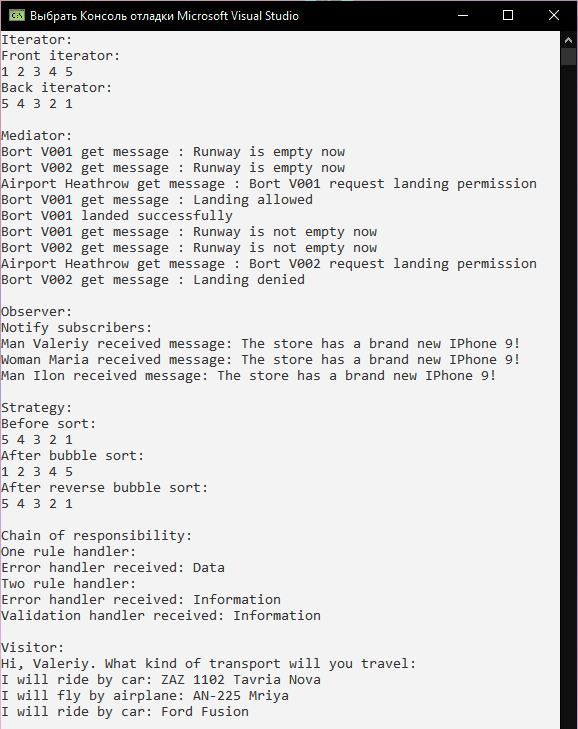


Рисунок 3.1 – Резульати виконання програми

Висновки

Протягом виконання цієї лабораторної роботи ознайомився з видами шаблонів проектування ПЗ, вивчив шаблони поведінки та отримав базові навички із застосування шаблонів Ітератор (Iterator), Посередник (Mediator), Спостерігач (Observer), Стратегія (Strategy), Ланцюг обов'язків (Chain of Responsibility), Відвідувач (Visitor).

Створив програму, з використанням шаблонів поведінки мовою програмування C#.