v

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №**5

з дисципліни “Основи об’єктно-орієнтованого програмування”

тема “ Реалізація складних поведінкових шаблонів проектування ”

|  |  |  |
| --- | --- | --- |
| Виконала  студентка II курсу  групи КП-83  Дзьоба Уляна Олегівна  (*прізвище, ім’я, по батькові*)  варіант №5 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 2020 р.  викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

Київ 2020

**Мета роботи**

Ознайомлення з основними характеристиками шаблонів «Ланцюжок обов’язків», «Команда» та «Посередник», запам’ятовування поширених ситуацій, коли використання цих шаблонів є доцільним, набуття вмінь та навичок реалізації шаблонів під час створення програмного коду.

**Постановка завдання**

1. Під час надходження замовлення на кухню ресторану його послідовно обробляють декілька кухарів. Наприклад, один кухар бере на себе відповідальність за приготування салату та передає замовлення наступному кухарю, який може відповідати за приготування м’ясної страви тощо. Таким чином, замовлення обходить всіх кухарів, які з нього вибирають ту частину блюд, яка лежить у зоні їх відповідальності. За допомогою шаблону проектування реалізувати механізм обробки замовлення кухарями ресторану.
2. Програмні засоби віддаленої роботи з певною інформаційною системою надають можливість зайти до іншого клієнта системи та, ввівши потрібний пароль, віддавати команди на виконання іншому клієнту, запущеному на іншій машині. За допомогою шаблону проектування реалізувати механізм пересилання команд на виконання до іншого клієнта (Необов’язково створювати мережеву версію програми. Достатньо імітувати роботу двох клієнтів одразу).

**Вибір шаблону**

Для розв’язання першої задачі був обраний поведінковий шаблон проектування «Ланцюжок обов’язків» оскільки згідно з теоретичними відомостями він використовується коли повідомлення в системі оброблюються за схемою “оброби сам або передай іншому”, що відповідає поставленій задачі, в якій кухарі послідовно готують і передають страви.

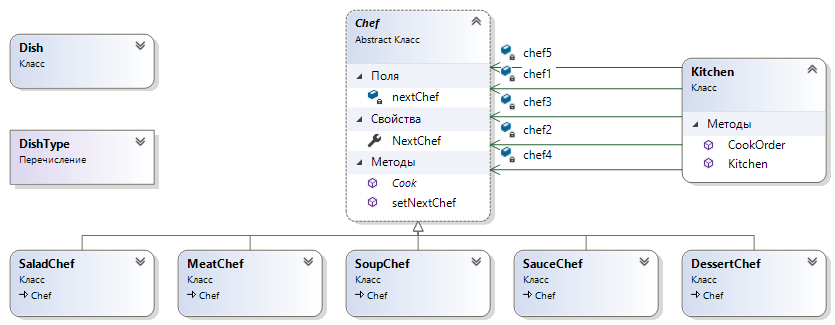
**Текст програми**

|  |
| --- |
| **Програмний код** |
| class Program  {  static void Main(string[] args)  {  List<Dish> order = new List<Dish> { new Dish("guacamole", DishType.Sauce), new Dish("tomato soup", DishType.Soup),  new Dish("steak", DishType.MeatDish), new Dish("tiramisu", DishType.Dessert), new Dish("chiken shnitzel", DishType.MeatDish),  new Dish("caesar", DishType.Salad), new Dish("caprese", DishType.Salad)};  Console.Write("Order: ");  foreach(Dish d in order)  {  Console.Write(d.Name+" ");  }  Console.WriteLine("");  Kitchen kitchen = new Kitchen();  kitchen.CookOrder(order);  Console.ReadKey();  }  }  public abstract class Chef  {  private Chef nextChef;  public Chef NextChef { get => nextChef; set => nextChef = value; }  public void setNextChef(Chef nextChef)  {  this.nextChef = nextChef;  }  public abstract void Cook(List<Dish> order);  }  class SaladChef:Chef  {    public override void Cook(List<Dish> order)  {  List<Dish> remainedOrder = new List<Dish>();  foreach (Dish dish in order)  {  if (dish.Type == DishType.Salad)  {  Console.WriteLine("Cooked {0}", dish.Name);  }  else  {  remainedOrder.Add(dish);  }  }  if (remainedOrder.Any())  {  if (this.NextChef != null)  {  this.NextChef.Cook(remainedOrder);  }  else  {  throw new ApplicationException("Lacks next chef unrecognized dishes");  }  }  else  {  Console.WriteLine("Order is cooked");  }  }  }  class DessertChef:Chef  {  public override void Cook(List<Dish> order)  {  List<Dish> remainedOrder = new List<Dish>();  foreach (Dish dish in order)  {  if (dish.Type == DishType.Dessert)  {  Console.WriteLine("Cooked {0}", dish.Name);  }  else  {  remainedOrder.Add(dish);  }  }  if (remainedOrder.Any())  {  if (this.NextChef != null)  {  this.NextChef.Cook(remainedOrder);  }  else  {  throw new ApplicationException("Lacks next chef unrecognized dishes");  }  }  else  {  Console.WriteLine("Order is cooked");  }  }  }  class Dish  {  public string Name;  public DishType Type;  public Dish(string name, DishType type)  {  Name = name;  Type = type;  }  }  class Kitchen  {  IChef chef1 = new SaladChef();  IChef chef2 = new MeatChef();  IChef chef3 = new SoupChef();  IChef chef4 = new SauceChef();  IChef chef5 = new DessertChef();  public Kitchen()  {  chef1.setNextChef(chef2);  chef2.setNextChef(chef3);  chef3.setNextChef(chef4);  chef4.setNextChef(chef5);  }  public void CookOrder(List<Dish> order)  {  chef1.Cook(order);  }  }  } |

**Приклад результатів роботи програми**

|  |
| --- |
| **Вивід у консоль** |
|  |

**Діаграма класів**

****

**Вибір шаблону**

Для розв’язання другої задачі був обраний складний поведінковий шаблон проектування «Команда» оскільки згідно з теоретичними відомостями він оброблює команди у вигляді об’єкту і дозволяє предавати їх інщим об’єктам, що і вимагається в умові задачі.

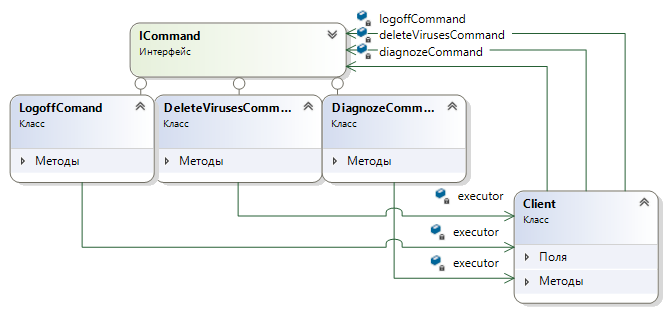
**Текст програми**

|  |
| --- |
| **Програмний код** |
| **class Program**  **{**  **static void Main(string[] args)**  **{**  **Client client1 = new Client("annsmith", "password1");**  **Client client2 = new Client("milkyway", "password2");**  **client1.Connect(client2, "password2");**  **client1.SendDeleteVirusesCommand();**  **client1.SendLogoffCommand();**  **Console.WriteLine("");**  **client2.Connect(client1, "password1");**  **client2.SendDeleteVirusesCommand();**  **client2.SendDiagnozeCommand();**  **Console.ReadKey();**  **}**  **}**  **internal class Client**  **{**  **private string password;**  **public string Username;**  **ICommand logoffCommand;**  **ICommand deleteVirusesCommand;**  **ICommand diagnozeCommand;**  **Client currentConnected;**  **public Client(string username, string password)**  **{**  **Username = username;**  **this.password = password;**  **}**  **public void Connect(Client client, string password)**  **{**  **if (client.CheckPassword(password)){**  **this.logoffCommand = new LogoffComand(client);**  **this.deleteVirusesCommand = new DeleteVirusesCommand(client);**  **this.diagnozeCommand = new DiagnozeCommand(client);**  **currentConnected = client;**  **Console.WriteLine("{0} connected to {1}", this.Username, client.Username);**  **}**  **else**  **{**  **Console.WriteLine("Password is incorrect");**  **}**  **}**  **public void SendLogoffCommand()**  **{**  **if (this.logoffCommand != null)**  **{**  **Console.WriteLine("Sending logoff command to {0}", currentConnected.Username);**  **this.logoffCommand.Execute();**  **}**  **else**  **{**  **Console.WriteLine("Connect first");**  **}**  **}**  **public void SendDeleteVirusesCommand()**  **{**  **if (this.deleteVirusesCommand != null)**  **{**  **Console.WriteLine("Sending delete viruses command to {0}", currentConnected.Username);**  **this.deleteVirusesCommand.Execute();**  **}**  **else**  **{**  **Console.WriteLine("Connect first");**  **}**  **}**  **public void SendDiagnozeCommand()**  **{**  **if (this.diagnozeCommand != null)**  **{**  **Console.WriteLine("Sending diagnoze command to {0}", currentConnected.Username);**  **this.diagnozeCommand.Execute();**  **}**  **else**  **{**  **Console.WriteLine("Connect first");**  **}**  **}**  **public Boolean CheckPassword(string password)**  **{**  **if (this.password == password)**  **{**  **return true;**  **}**  **else return false;**  **}**  **public void Logoff()**  **{**  **Console.Write(this.Username+": ");**  **Console.WriteLine("Logged off account");**  **}**  **public void DeleteViruses()**  **{**  **Console.Write(this.Username + ": ");**  **Console.WriteLine("All viruses deleted");**  **}**  **public void Diagnoze()**  **{**  **Console.Write(this.Username + ": ");**  **Console.WriteLine("System is diagnozed");**  **}**  **}**  **interface ICommand**  **{**  **void Execute();**  **}**  **class LogoffComand:ICommand**  **{**  **Client executor;**  **public LogoffComand(Client c)**  **{**  **executor = c;**  **}**  **public void Execute()**  **{**  **executor.Logoff();**  **}**  **}**    **class DeleteVirusesCommand:ICommand**  **{**  **Client executor;**  **public DeleteVirusesCommand(Client c)**  **{**  **executor = c;**  **}**  **public void Execute()**  **{**  **executor.DeleteViruses();**  **}**  **}**  **class DiagnozeCommand:ICommand**  **{**  **Client executor;**  **public DiagnozeCommand(Client c)**  **{**  **executor = c;**  **}**  **public void Execute()**  **{**  **executor.Diagnoze();**  **}**  **}** |

**Приклад результатів роботи програми**

|  |
| --- |
| **Вивід у консоль** |
|  |

**Діаграма класів**

****

**Висновки щодо роботи програм**

Виконавши дану лабораторну роботу було створено дві програми мовою C# що демонструють ознайомлення з використанням шаблонів проектування. Шляхом аналізу заданих ситуацій і порівняння їх характеристик з характеристиками стандартних зразків використання запропонованих шаблонів проектування для реалізації програм було обрано шаблони «Ланцюжок обов’язків» та «Команда» і показано доцільність цього вибору шляхом демонстрації роботи програм.

Розробка велася за допомогою Microsoft Visual Studio.